Anne Jonker (Student number: 2528666)

Master Thesis

-

Bag & Tag'em, A new Dutch stemming algorithm

in Exact Sciences

August 1, 2019

Vrije Universiteit Amsterdam

Faculty of Exact Sciences

Author: Anne Jonker (Student number: 2528666)

Contact information: annejonker@gmail.com

Supervisors: Jornt de Gruijl (Bright & Company), Corné de Ruijt (VU), Prof. Dr. Sandjai Bhulai (VU), and Prof. Dr. Guszti Eiben (VU)

Title: Master Thesis - Bag & Tag'em, A new Dutch stemming algorithm

Research period: January 2019 - August 2019

Report date: August 1, 2019

Presentation date: August 15, 2019

Presentation location: Kerkweg 31, 3603 CL Maarssen, The Netherlands

Study line: Master Business Analytics

University address: De Boelelaan, 1081a, 1081 HV Amsterdam, The Netherlands

Bright & Company address: Kerkweg 31, 3603 CL Maarssen, The Netherlands

Page count: 72+10

Abstract: The aim of this thesis is to overcome the problems that the current state-of-the-art stemmers face in the Dutch language. The main issue is that the current stemmers cannot handle 3rd person singular forms of verbs, wherein the verbs ends with a *t*, as well as many irregular words and forms. This is done by combining a new tagging module with a new stemmer that uses a set of rigid rules: the Bag & Tag'em (BT) algorithm. The tagging module is tested on and evaluated using three algorithms: *MLR*, *NN*, *XGB*. The stemming module is compared with and evaluated to current state-of-the-art stemming algorithms for the Dutch Language. Even though there is still room for improvement, the new BT algorithm performs well in the sense that it is more accurate than the current stemmers and faster than brute-force-like algorithms.

Keywords: Text mining, Stemming, Tagging, Dutch language

Preface

Before you lies the thesis: "Bag & Tag'em - A new Dutch stemming algorithm". It has been written to fulfil the final graduation requirements of the Business Analytics program at the Vrije Universiteit Amsterdam (VU). I was engaged in researching and writing this Master thesis from February to August 2019.

Writing this thesis has been a journey that lasted well over six months and I could not have completed it without the help from a lot of people. First of all I would like to thank Jornt De Gruijl from Bright & Company who came up with the idea of a new stemmer. In addition to the idea itself, I am also grateful for the continuous support in programming the algorithm, and helping me tackle the numerous problems that arose along the way. Furthermore, Bright & Company made me feel at home from the very first moment I entered the office in Maarssen, the Netherlands. I had a great time with all of my colleagues while working there!

Another person I am very grateful to have met was Corné de Ruijt, who was my University supervisor. Corné assisted in tackling theoretical obstacles and in something that should not be forgotten in a project this size: time management. Furthermore, I want to thank my other supervisors: Prof. Dr. Sandjai Bhulai and Prof. Dr. Guszti Eiben for taking the time to read and correct my thesis.

In addition, I would like to thank my friend Tjalling Otter for helping to manually check the Dutch documents and proof-reading chapters for grammar, and finally my wonderful girlfriend Janine Nanlohy who supported me endlessly throughout the conception of this thesis.

It is my sincere wish that you enjoy reading this thesis, and that it may inspire future research as well as assist in decision-making challenges.

Anne Jonker, August 1, 2019

The Author

List of Figures

List of Tables

Contents

1 Introduction

1.1 Introduction

With the current availability of large quantities of digitised documents and data, the data mining research field faces a shift from a lack of available data to an abundance of it. The idea of data mining is to discern valuable patterns in data, which are not or cannot be perceived by even the most observant analysts. The field has evolved tremendously over the last decade and is now entering a mature phase (Weiss, Indurkhya, & Zhang, 2015).

The increased availability of data has also resulted in a resurgence of a sub-field of data mining: text mining. In order to perform analysis on unstructured data such as text, data is commonly preprocessed in a number of ways. One such commonly used preprocessing step is to reduce the number of variants of the same word back to a common root in order to reduce noise and enhance the accuracy of the analyses that follow. This is referred to as *stemming*, whereby words are reduced to their stem. Stemming has been studied thoroughly for the English language and considerable success has already been achieved. This is mainly done through use of the Porter algorithm (Porter, 2001) that uses a rigid rule-based system to stem a word.

The problem with many other languages, in particular the Dutch language, is that they have been less of a focal point for research and that many words are irregular when it comes to their derivations. As such, they do not conform to the otherwise established rule sets, forming exceptions. The goal of this thesis is to construct a new algorithm that is more accurate than the state-of-the-art algorithms currently in use for the Dutch language. The algorithm will make use of two important components.

The first component will be a new tagging algorithm that categorises a word based on the letters that comprise it. This differs from established tagging algorithms which make use of large corpora of pre-tagged words. In theory, this would decrease the requisite computational resources, at the cost of some degree of accuracy, but more robustness in coping with e.g. neologisms.

The second component will be to apply pre-determined rules to stem a word based on the aforementioned tag. This part of the approach is similar to the Porter algorithm, but will instead use rules specific to the Dutch language.

The final algorithm could aid in analysing large quantities of Dutch text documents, due to its robustness and computational efficiency compared to current standards. As such, many existing and novel applications of text mining stand to gain from the algorithm described in this article.

Text mining is currently already implemented in numerous fields. It can be applied in sentiment analyses to determine the overall reputation of a company (Pang, Lee, et al., 2008), identifying and analysing cybercrime activities (Kontostathis, Edwards, & Leatherman, 2010) or fraud detection (Phua, Lee, Smith, & Gayler, 2010). These are just a few examples of the increasing possibilities to apply text mining to numerous problems faced by companies and researchers on a daily basis.

1.2 Background Dutch language

Before the various methods in text mining are explained, first some background information is given on the Dutch language. First the various types of words in Section 1.2.1 afterwards grammar rules in Section 1.2.2, which will be referred back to in Chapter 3. Section 1.2.2 will also show the problems some of the current state of the art stemmers have when it comes to verbs, which is the main reason why a new stemmer had to be build.

1.2.1 Type of words

In the Dutch language there are two ways to categorise types of words in a sentence. One way is called : *taalkundig ontleden* (linguistic parsing) the other: *redekundig ontleden* (reasonably dissect).

Redekundig ontleden groups words together that belong together. For example *De lange vrouw wilde een limonade bestellen* (The tall woman wanted to order a lemonade) splits up the sentence in three parts: *De lange vrouw* (The tall woman) which is the subject of the

sentence, *wilde bestellen* (wanted to order) which are the verbs explaining what the subject is doing, and *een limonade* (a lemonade) which explains what the subject wanted.

For reasons that will be described later on in this thesis, only the *taalkundig ontleden* (linguistic parsing) will be explained in more detail. In linguistic parsing each word is labelled (tagged) based on the type of word in the sentence.

In total are there 22 word types in Dutch linguistic parsing shown in Table 33 in Appendix A. Not all of the word types will be used in this thesis, however the most important word types for this thesis are described in Table 1. The Auxiliary verb, Linking verb and Independent verb are combined in one general word type: *Werkwoord* (verb).

Type	Translation	Example	Translation
Lidwoord	Article	De lange vrouw	The tall woman
Zelfstandig naamwoord	Noun	De lange vrouw	The tall woman
Bijvoeglijk naamwoord	Adjective	De lange vrouw	The tall woman
Werkwoord	Verb	Ik zal lopen	I shall walk
Persoonlijk voornaamwoord	Personal pronoun	Ik zal lopen	I shall walk
Voorzetsel	Preposition	Ik geef dit aan haar	I give this to her
Bijwoord	Adverb	Morgen komt hij niet	He will not come tomorrow

Table 1. Overview most important word types *taalkundig ontleden* in the Dutch language

1.2.2 Grammar rules

To understand why certain rules are applied for stemming, there is a necessity to know how Dutch words are constructed. First the verbs will be explained, afterwards the adjectives and finally the nouns. The other word types are less interesting due to the fact that they are most likely to be stop words which is explained in Section 2.1.3 or not able to be stemmed at all.

In the Dutch language there are ten forms of time in verbs. For each verb (except for the

past participle and Imperative), there are six personal forms. The six personal pronouns determines who did something and are presented in Table 2, together with their respective translation.

Personal pronoun	Translation
ik	I
jij	you
hij / zij	him / her
wij	us
julllie	them
zij	they

Table 2. Personal pronouns in the Dutch language

The ten forms of time as presented in Table 34 in Appendix B and it shows the first person of the regular verb *leren* (to study). The Table shows that the number of forms when only the word *leren* is taken into account, for the singular form can be reduced to four forms: *leer*, *leren*, *geleerd*, *leerde*. The other forms are created by using *hulp werkwoorden* (auxiliary verbs) to determine the time and if you shall do it. This is crucial for the rest of this thesis to understand why only four types of verbs are examined, instead of the ten. Therefore only the *ott*, *vtt*, *ovt* will be discussed since *leren* is an infinitive which is also contained in the *ott*.

First the *Onvoltooid tegenwoordige tijd (ott)* (Present tense) is explained using three verbs, to deduct the grammar rules. In general the difference with stem works as a basic rule, but when the column of *weten* (to know) is observed then two strange things occur. No additional *t* is added to the stem because in the Dutch language there are no verbs that end with double *t* or *d*. Furthermore in the plural form an *e* is removed in the stem. *Weten* is therefore an irregular verb and many examples like this are present in the Dutch language. For now we can assume that the general rules work rather well. Due to this assumption other state of the art stemming algorithms fail on these types of verbs. Especially if or when not to remove the letter *t* at the end of a verb.

Personal pronoun	werken (to work)	worden (to become)	weten (to know)	Difference with stem
ik	werk	word	weet	stem
jij	werkt	wordt	weet	stem + t
hij	werkt	wordt	weet	stem + t
wij	werken	worden	weten	stem + en
jullie	werken	worden	weten	stem + en
zij	werken	worden	weten	stem + en

Table 3. Example of verb forms in the Present Simple (ott)

The Present perfect (vtt) as shown in Table 4, shows that *werken* is not suffix stripped the same as the two other verbs. Where simply removing the prefix *ge* from the verbs *worden* and *weten* results in the infinitive, this is not the case for *werken*. The helping verb is different between *werken* and *weten* on one side (helping verb is *hebben* (to have) and *worden* (helping verb is *zijn* (to become). This does not have an influence on how the main verb is constructed.

Personal pronoun	werken (to work)	worden (to become)	weten (to know)	Difference with stem
ik	heb gewerkt	ben geworden	heb geweten	ge- & -t or -en
jij	hebt gewerkt	bent geworden	hebt geweten	ge- & -t or -en
hij	heeft gewerkt	is geworden	heeft geweten	ge- & -t or -en
wij	hebben gewerkt	zijn geworden	hebben geweten	ge- & -t or -en
jullie	hebben gewerkt	zijn geworden	hebben geweten	ge- & -t or -en
zij	hebben gewerkt	zijn geworden	hebben geweten	ge- & -t or -en

Table 4. Example of verb forms in the Present perfect (vtt)

The Simple past tense (ovt) as shown in Table 5 shows a more complicated deduction of "simple" rules, grammar should normally follow. In general one can remove either a *te(n)* or *de(n)* from the Past tense form, to convert the verb into present simple. The additional *n* is stripped in case the verb is a plural. For these verbs (almost all) the singular personal share the same affixes among the singular forms, which is also true for the plural personal forms.

Personal form	werken (to work)	worden (to become)	weten (to know)	Difference with stem
ik	werkte	werd	wist	-te or new word
jij	werkte	werd	wist	-te or new word
hij	werkte	werd	wist	-te or new word
wij	werkten	werden	wisten	-ten or -en
jullie	werkten	werden	wisten	-ten or -en
zij	werkten	werden	wisten	-ten or -en

Table 5. Example of verb forms in the Past tense (ovt)

All these various forms of verbs and when to remove certain parts of a word or even adding vowels is a huge challenge, which is tried to be tackled in this thesis. The verbs are the most challenging part but also the adjectives are not always as easy to stem as it first seems to be.

Similar to the English language an adjective is generally placed between an article and a noun, since the adjective describes something about the noun. For example *de mooie bloem* (the beautiful flower). Most of the adjectives have the same suffixes as shown as an example in Table 6. These are the superlatives and again there are irregular adjectives (*goed*), which is the same as in English (good, better, best). Similar with the previous examples of the verbs *weten* the number of vowels in the adjective *rood* also changes.

Similar problems with the adjectives occur in the verbs but the number of various forms is smaller and therefore easier to capture in rules.

Mooi (beautiful)	Goed (good)	Rood (red)	Difference with stem
mooi	goed	rood	stem
mooie	goede	rode	-e
mooier	beter	roder	-er or new word
mooist	best	roodst	-st or new word
mooiste	beste	roodste	-ste or new word

Table 6. Adjective stemming of superlatives

For nouns there are two important features to help reducing the noun to its singular form. The first is more obvious when it comes to plural nouns. The second feature are the *verklein-woorden* (diminutives). To reduce a plural noun to a singular noun only a few suffixes has to be removed: *-en*, *-s*, or more uncommon but still relevant :*eren*.

For the diminutives either the suffix *-tjes* for plural or *tje* for singular, is removed. Again there are exceptions, which will need a slightly different approach to be stemmed correctly. The technique on how to get the correct stem will be discussed in Chapter 3.

2 LITERATURE

2.1 Text mining

Text mining is the process of gathering and extracting useful information from unstructured textual data sources (Feldman & Sanger, 2007). This is accomplished by identifying patterns in a collection of documents, which results in such things as identifying key concepts and keywords.

Text mining shares many high-level architectural aspects with data mining. These include preprocessing and pattern-exploring techniques. Many of the pattern-exploring techniques used in text mining share their origins with those in the general data mining field (Feldman & Sanger, 2007).

Differences mainly arise due to the often required preprocessing of text data. Whereas in data mining data the data is more frequently structured (i.e. in tabular form), this is not necessarily the case for text mining (Feldman & Sanger, 2007). In order to establish structure in the unstructured data preprocessing is an important step. During preprocessing, text mining tends to focus mainly on identifying, integrating and normalising representative features in documents. This results in data that analyses can be conducted on.

The text mining process itself can be summarised in seven steps, as described in the next parts of this literature study. The steps described are not all necessary in the text mining process, but are considered standard practice. The procedure is presented in Figure 1 and illustrates the preprocessing part, as this is the focal point of this thesis:

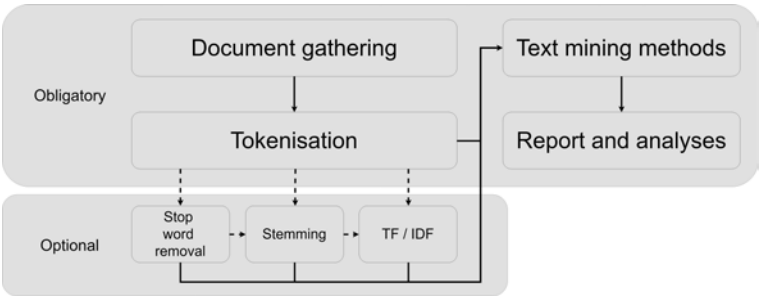

Figure 1. Text mining process adapted from (Vijayarani, Ilamathi, & Nithya, 2015)

2.1.1 Gathering of documents

Development and testing of a stemming algorithm starts with a representative selection of words or text in the target language. For our approach, we opted to use a fairly large corpus of documents in the Dutch language.

There are many ways to gather documents: either from extant document warehouses, databases or data lakes. The benefit of collecting documents from these sources is that it is likely that they have already been cleaned, and that their quality is likely to be adequately high. A downside of using these sources, is that in general these databases were constructed for a specific goal. Therefore it might lack some information required for the intended analyses.

Another method is to utilise web crawlers that automatically retrieve documents from the internet (Weiss, Indurkhya, Zhang, & Damerau, 2010). The benefit of using a web crawler is to gather information fast from many various sources. The downside of doing so is that - despite the potentially high availability of relevant documents - their quality is often considered to be lacking or the data lineage is unclear. This may necessitate an arduous and lengthy cleaning process, and is only to be considered when other available databases are lacking or even unavailable.

After the documents are collected they have to be standardised to the same format. The reason for this is that the original documents were likely generated in various ways. Examples of this are documents generated by a word processor, ASCII (i.e. plain-text) sources, or even pictures of documents. Currently, a best practice in the text mining field is to save documents in the the XML format, which uses tags to divide a document into separate parts.

The benefit of the XML format is that not only the text can be stored and retrieved, but also metadata such as the associated subject, authors and publication dates. This facilitates mining algorithms when clustering topics, or can be used to assign priority to recent sources, as they are likely to be more important.

2.1.2 Tokenisation

Tokenisation is the process of dividing strings of text into separate tokens. These tokens can be symbols, words, terms, or some other type of meaningful elements (Vijayarani, Janani, et al., 2016). Tokenisation generally follows a few simple heuristics, explained in detail by (Weiss et al., 2015):

1. a new token starts with a space, tab or the end of a line. These are called whitespaces.
2. characters [( ) < > > ? ;] are always delimiters and therefore are treated as tokens.
3. characters [. ,] could either be delimiters (and therefore tokens) or not, depending on the characters around them. An example of this is that the number 44,210.98 is one number and should not be treated as 44, 210, and 98. Therefore the heuristic checks whether the previous and next character are numbers.

2.1.3 Stopword removal

After tokenising the documents, certain so-called stopwords are typically removed. These are words that are so common that they convey little to no information. Examples are: *de/het (the), een (a/an), en (and), deze (this), (dat) that*, etc. This is done to save on computational time, as such words do not have to be stemmed or analysed.

The stopword removal algorithm simply scans the documents for occurrences of these elements with the use of a comprehensive list. That list is constructed from the most frequently-used words that occur in a large set of documents. For the Dutch language, such a list is already available and contains approximately 150 words (Porter, 2001).

In the case of sequential models, which will be explained in Section 2.4, stopwords are not removed. In that case, the word itself may not offer any novel information, but the sequence that it is embedded in could.

2.1.4 Term Frequency Inverse Document Frequency

Term Frequency Inverse Document Frequency (TF/IDF) is a technique used in text mining, as a weighting factor for features. The general idea is that the importance of a word increases as the word frequency in a document increases, but it is offset by a weight indicating how common that word is, as found in reference documents (Salton & Buckley, 1988).

IDF by itself can also be used to remove the importance from common words, that have barely any information (Paik, 2013). Therefore TF/IDF can be used for: relevance ranking, scoring and creating a stop word list, as explained in Section 2.1.3.

The calculation consists of two components: The Term Frequency (TF) and the Inversed Document Frequency (IDF) (Aizawa, 2003).

The TF is the count of term t in document d, where TF is defined as: $\mathrm{TF}(t,d) = f_{t,d}$.

The IDF as shown in Equation (2.1) is the measure on how much information is gained from t in the document collection D. N stands for the total number of documents and is divided by the number of documents in the dataset that contain t. The logarithm is used to dampen the effects of the IDF function.

$$\mathrm{IDF}\,(t,D) = log\left(\frac{N}{|\{d \in D : t \in d\}|}\right) \tag{2.1}$$

The two components multiplied give the TF/IDF function, as presented in Equation (2.2).

$$\mathrm{TF/IDF}\,(t,d,D) = \mathrm{TF}\,(t,d) \cdot \mathrm{IDF}\,(t,D) \tag{2.2}$$

2.2 Stemming

Once separate words are identified and stopwords are removed, documents typically have to be stemmed. Stemming is the process by which words or grammatical forms are reduced to common stems (Jivani et al., 2011). An example of this is to reduce the words *walking* and *walked* to their common stem: *walk*. The purpose of doing so is to decrease the number of

occurrences of words that have different forms, yet convey the same meaning.

Several stemming algorithms are applied in the text mining field, and can be categorised by one of four classes (Jivani et al., 2011):

1. Table Methods: Production technique. This stemming algorithm relies on a large table where the stem is found based on the word.
2. Truncating Methods: Lancaster, Porter, Snowball. These stemming algorithms only use affix stripping.
3. Inflectional and Derivational Methods: Part of Speech, Sequential models. These stemming algorithms rely more on large corpora and context information.
4. Statistical Methods: Hidden Markov Models (HMM) . The stemming algorithms use probabilities to determine what the correct stem should be, mostly done in forms of Neural Networks.

2.2.1 The production technique

The production technique works in a manner that is antipodal from that of the other techniques. It generates a look-up table from all stems found in the Lexicon of the language. Utilising the same example as before, *walk* will prompt the program to generate a table with all possible grammatical derivations of the stem. This look-up table is shown in Table 7.

Stem	Form
Walk	Walk
Walk	Walks
Walk	Walked
Walk	Walking

Table 7. Example of production technique (English)

The problem is that languages change over time. New words and different spellings are added to dictionaries, which implies the need for constant maintenance of the look-up tables. As this is likely to be a manual process, there is also a chance of introducing human errors. The handling of neologisms requires a robust approach based on generalisations.

Another problem with the production technique is the large amount of computational power required to search through the entire table for every word. This is especially true for a language that, as stated, presents a large number of grammatical irregularities and exceptions, like Dutch. The Dutch translation of walk is "*loop*". All its different derivations are shown in Table 8. This example illustrates that the size of the look-up table has already doubled, even when only a single verb is considered. This combined with the ever expanding size of the dictionaries poses a problem.

Stem	Form
Lop	Loop
Lop	Loopt
Lop	Lopen
Lop	Liep
Lop	Liepen
Lop	Gelopen
Lop	Lopende

Table 8. Example of production technique (Dutch)

2.2.2 Prefix and Suffix-stripping algorithms

A more common Stemming technique is Prefix and Suffix-stripping. This technique relies on a set of rules in order to remove certain parts of words, be it at the start (Prefix) or at the end (Suffix). A few basic rules (of many) are as follows (Willett, 2006):

1. If the word ends with -ing, remove ing,
2. If the word ends with -ed, remove ed,
3. If the word ends with -s, remove s.

For the example given in Table 7, this set of rules would work perfectly. However, problems occur when irregular verbs are concerned; for example *run* and *ran*. Both have the same stem - run - but this technique would identify two different stems instead. The Dutch language contains many irregular verbs and nouns and is therefore problematic in this regard. Every language has its own characteristics and will need a language specific stemmer for each one.

There are a few different variants of stemming algorithms that serve a specific purpose. Some require the stripped stem to be a word that is contained in the Lexicon of the language (akin to the process lemmatisation or reducing a word to canonical form). If this is not the case then a new rule has to be applied to reduce it even further to derive a proper stem, or expand the Lexicon itself.

The benefit of rule-based stemming techniques is fast conversion of documents (Jivani et al., 2011), but the downside is potentially large inaccuracies due to the absence of language-specific rules.

The most common algorithms for the prefix and suffix stripping are: Porter (Willett, 2006), Lancaster(Paice, 1990) and Snowball (Porter, 2001). Snowball is a version of the Porter algorithm adapted to make the set of rules more language-specific.

2.3 Part of Speech Tagging

A different approach to stemming is to make use of Part of Speech Tagging (PoS), in which words are tagged based on their function in the sentence and then accordingly handled further by different sets of rules (Monz & De Rijke, 2001).

The tagger tags words as a type or "speech tag", i.e.: noun, adjective, verb and so on. A rule-based approach based on probability was proposed by (Brill, 1992). The tagger starts by assigning each a tag to each word, by estimating the probability obtained from a large previously tagged corpus. This is done disregarding any context given by words in the same sentence. The example that was presented in the paper shows that in the following two sentences the word *run* is tagged as a verb, since the word *run* was most likely to be a verb in the tagged corpus, despite it not being a verb in the first sentence.

1. The *run* lasted thirty minutes.
2. We *run* three miles every day.

The tagger's performance improved further by implementing two procedures. The first procedure is that words that were not contained in the training corpus and started with a capital letter were tagged as a noun. This resulted in improved performance when names were

involved, specifically.

The second procedure also took the words not contained in the training corpus and analysed the last three letters of the word. The different three-letter combinations were then compared to other words in the trained tagged corpus, to determine a probability based on similarities of words with the corresponding tags.

The conclusion of the paper showed that the performance of this simple part-of-speech tagger was roughly the same as other stochastic taggers, but had more advantages. These include speed, applicability and easier transferable in different languages.

Two PoS taggers for the Dutch language are Alpino (Van der Beek, Bouma, Malouf, & Van Noord, 2002) and Frog (Bosch, Busser, Canisius, & Daelemans, 2007) which also use a large pre-tagged corpus from the CELEX database (Van der Wouden, 1990).

To achieve automated PoS tagging, many different algorithms or methods are available. To maintain brevity and overview, three methods are examined and explained in the next sections. Since PoS tagging is a multinomial classification problem various algorithms can be applied ranging from relatively simple to more complex.

2.3.1 Multinomial Logistic regression (MLR)

The Multinomial Logistic regression is one of the most basic regression techniques for multinomial classification problems (He & Zelikovsky, 2006). MLR is a classification method which is a generalised form from the binary logistic regression method. Like any regression, it requires independent and dependent variables.

One of the assumptions made for the model is that each dependent variable (tag) has only one feature for each observation. This means that in the example described previously the word *ren* can only have one tag. Either as a verb (*run*) or a noun (*rabbit cage*) but not both. This assumption on the one hand simplifies the problem but also might lose information about the correct stem (Greene, 2003).

The MLR makes a linear combination to determine which class (k) belongs to observation (i). In the case of word tagging the class would be the tag and the observation the word. The

regression coefficients are described by $\beta_{m,k}$ for features $m \in \{1,\ldots,M\}$ and $k \in \{1,\ldots,K\}$. The probability function is described in Equation (2.4), and the log-likelihood that is used in MLR is shown in Equation (2.5).

Scikit-learn (Pedregosa et al., 2011) provides multiple parameters that can be adjusted to further improve the model. Not all will be discussed, but the parameters that were optimised and what they do are.

Since tagging is not a binary problem but a multinomial, as there are more than two possible tags, the parameter of *multi_class* was set to *multinomial*. There are multiple algorithms that can be used in the multinomial logistic regression for optimisation. Each algorithm has advantages and disadvantages depending on the dataset. The algorithms that are tested are: *newton-cg*, *lbfgs*, *sag* and *saga*.

The reason for not using the *liblinear* algorithm is that it is limited to one-versus-rest schemes, while the other algorithms are capable of handling multinomial loss. The Equations: (2.4), (2.5), (2.6) and derivatives were obtained from (Hastie, Tibshirani, Friedman, & Franklin, 2005), which are needed to explain the workings of the optimisation algorithms and their respective penalty functions.

Newtons Method for optimisation (Newton-cg) is an second order optimisation function to determine either a maximum or minimum of a function $f(\beta_1, \beta_2, ..., \beta_p)$. It is an iterative function as described in Equation (2.3), where $\bigtriangledown f(\beta^{(n)}$ is the gradient of $f(\beta^{(n)}$ and H the Hessian of $f(\beta^{(n)}$, which is the matrix of second partial derivatives (Schmidt, Fung, & Rosales, 2007).

$$\beta^{(n+1)} = \beta^{(n)} - H^{-1}\left(\beta^{(n)}\right) \bigtriangledown f\left(\beta^{(n)}\right) \tag{2.3}$$

There are two main drawbacks of using Newtons method, whereas this method is computationally expensive due to the Hessian inverse matrix calculations (unless it is a diagonal matrix) and is attracted to saddle points. Newton-cg is capable of handling L_2 penalty functions, which will be explained in more detail further on in this section.

The limited Memory Broyden-Fletcher-Goldfarb-Shanno algortihm (lbfgs), is an optimisa-

tion algorithm belonging to the quasi-Newton method family, which uses a limited amount of computer memory. Similar to the newton method lbfgs uses an estimation to the inverse Hessian matrix to determine a minimum or maximum of a function. The difference lies in the fact that the inverse Hessian matrix is not stored, but stores the past k updates of the position $\beta^{(n)}$ and the gradient $\bigtriangledown f\left(\beta^{(n)}\right)$ (Zhu, Byrd, Lu, & Nocedal, 1997). This results in a more efficient usage of memory and therefore an increased computational speed . The lbfgs algorithm from Sklearn (Pedregosa et al., 2011) supports the L_2 penalty function.

Stochastic Average Gradient (SAG) optimises the sum of a finite number of smooth convex functions. The SAG iterations costs are independent from the number of terms in the sum, which is the same in other stochastic gradient techniques. The reason why SAG has a faster convergence rate compared to other black-box stochastic gradient techniques is that SAG stores the values of the previous m calculated gradient values, instead of storing all.(Schmidt et al., 2007).

The drawback of using SAG is the fact that it can only handle L_2 penalisation, in the package from Sklearn (Pedregosa et al., 2011). A variation on SAG is called SAGA, which overcomes the drawback of SAG, as SAGA can handle L_1, L_2, *elasticnet* penalty functions.

The algorithms are also tested on various penalty functions if the algorithm can handle these. The penalty functions are L_1, L_2, *elasticnet* or no penalty at all.

L_1 regularisation is also known as Lasso Regression and L_2 regularisation as Ridge Regression. In Lasso Regression an absolute value of magnitude of coefficient is added as penalty term to the loss function $||\beta_j||$. In Ridge Regression a squared magnitude of coefficients is added $||\beta||^2$.

In both Lasso and Ridge Regression a new variable λ is introduced, which determines how much of the penalty is added. If the value of λ is set to zero, the original loss function is retrieved. If the value of λ is set too high, too much penalty is added and leads to underfitting (Fu, 1998).

In Lasso the less important feature coefficients are reduced to zero. This removes certain features from the equation entirely, which works well if there are many features to select

from (Tibshirani, 1996). In Ridge regression, feature coefficients that obtain large values are penalised and therefore the coefficients are "shrunk". This helps in reducing model complexity, multi-collinearity and overfitting of certain features (Fu, 1998). Elastic net is a linear combination of Lasso and Ridge regression (Zou & Hastie, 2005). An additional parameter α is added to determine the weight on each of the penalty functions. If α is set to zero then the ridge regression is used as the penalty function and if α is set to one, the lasso regression is used. The penalty function is described in Equation (2.6) which is added to the log-likelihood function.

The number of maximum iterations until one of the solving algorithms converges, is also varied to ensure maximal performances.

$$\Pr(G = k|X = x) = \frac{e^{\beta_{0k}+\beta_k^T x}}{\sum_{\ell=1}^{K} e^{\beta_{0,l}+\beta_l^T x}} \tag{2.4}$$

$$\ell\left(\{\beta_{0k}, \beta_k\}_1^K\right) = -\left[\frac{1}{N}\sum_{i=1}^{N}\left(\sum_{k=1}^{K} y_{i\ell}\left(\beta_{0k+x_i^T\beta_k}\right) - log\left(\sum_{k=1}^{K} e^{\beta_{0k}+x_i^T\beta_k}\right)\right)\right] \tag{2.5}$$

$$+\lambda\left[(1-\alpha)\,||\beta||^2 + \alpha\sum_{j=1}^{p}||\beta_j||\right] \tag{2.6}$$

2.3.2 Neural Network (NN)

Literature suggests that Neural Networks show promising results in Natural Language Processing (NLP) (Schmid, 1994) and therefore the implementation of neural networks is investigated also.

NNs work with a black-box principle similar to how the human brain functions: it uses layers of nodes that each perform their own filtering and transformation of input and spread components of the task to be learned amongst themselves. There is a layer of input nodes, hidden layers filled with processing nodes and a final layer of output nodes, as shown in Figure 2. The workings of the Neural Network shown below are described in (Hansen & Salamon, 1990).

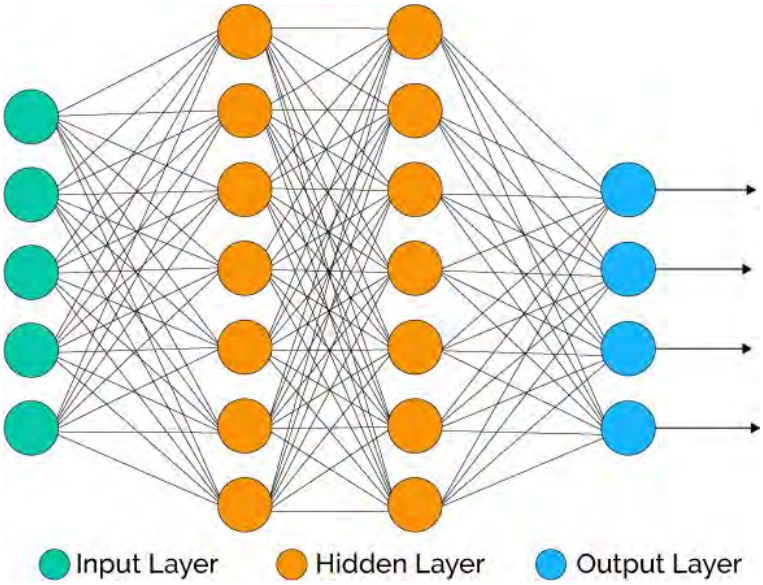

Figure 2. Neural Network obtained from (Rahul Bhatia, 2018)

The input layer contains all possible features, with each node directly representing a feature. If a feature is present in a word, then the node (or "neuron") gets activated and signals to the next hidden layer. The NN learns by propagating an error back from the output layer of the network to the input layer, adjusting weights of neurons based on their contribution to the final output.

This is done by adding up the squares of the differences of each of the errors output activation's and the value in the output neuron that is desired, which is the cost of a single training example. If the NN is accurate the cost will be relatively small, and high if the weights and biases have not been set properly. Therefore the average cost function on all of the training data in a NN has to be minimised which is done using gradient descent, which is the *learning* part in a NN also known as back propagation (Hinton, 1987).

It is important for the cost function to have a "smooth" output, where it is possible to find a local minimum by taking small steps in the gradient descent. This is a difference between artificial and biological neurons where in artificial neurons the activation ranges between 0 and 1, and biological neurons are binary (i.e. active or inactive).

In the next Equations (2.7) - (2.13) new parameters are introduced, which require some

explanation. The equations and their derivatives were obtained from (Nielsen, 2015). y_q is the desired output of neuron q, whereas $\alpha_q^{(L)}$ is the activation value of neuron q in the last layer L. Neuron p is the index of the neuron in the previous layer $(L-1)$ connecting to neuron q in the next layer L. C_0 is the cost function of a single neuron, which has to be minimised. The weight between neuron p and q is defined as $\beta_{pq}^{(L)}$ in layer L. The bias: $b_p^{(L)}$ is the bias of neuron p in layer L. The activation function f can be Sigmoid, ReLU or any other known activation function.

An overview of these parameters in a NN is presented in Figure 3, where for simplicity reasons and clarity, not all connections are drawn except for neuron p to neuron q.

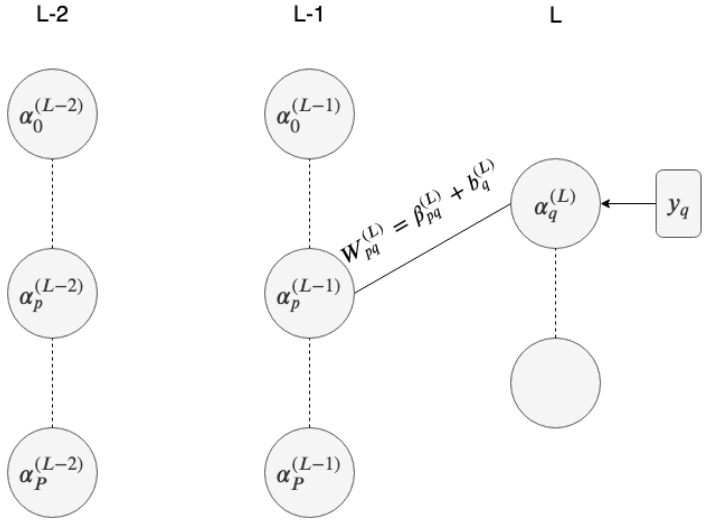

Figure 3. Overview of the parameters in a Neural Network

The weighted sum as described in Equation (2.7) is the sum of all the activation values in the previous layer multiplied by the respective weights to node q, to the total of which the bias (b_q) of node q is added.

$$W_q^{(L)} = b_q^{(L)} + \sum_{p=0}^{P} \beta_{pq}^{(L)} \alpha_p^{(L-1)} \tag{2.7}$$

The activation value α is explained in Equation (2.8), where f is a general activation function (i.e. Sigmoid, ReLU, tanh). Combining Equations (2.7) and (2.8) results in Equation (2.9), which shows the cost function for an output neuron which has to be minimised. This is done

by taking the squared difference between the activation value (α) of neuron q and the desired output $\left(y_q\right)$ of neuron q

$$\alpha_q^{(L)} = f\left(W_q^{(L)}\right) \tag{2.8}$$

$$C_0 = \sum_{p=0}^{P^{(L-1)}} \left(\alpha_q^{(L)} - y_q\right)^2 \tag{2.9}$$

The relative impact of change in weights, as presented in Equation (2.10) and relative impact of change in activation, as presented in Equation (2.11), on the cost function are derived using the chain rule for a single output neuron q.

$$\frac{\partial C_0}{\partial \beta_{pq}^{(L)}} = \frac{\partial w_q^{(L)}}{\partial \beta_{pq}^{(L)}} \cdot \frac{\partial \alpha_q^{(L)}}{\partial w_q^{(L)}} \cdot \frac{\partial C_0}{\partial \alpha_q^{(L)}} \tag{2.10}$$

$$\frac{\partial C_0}{\partial \alpha_p^{(L-1)}} = \sum_{p=0}^{P^{(L-1)}} \frac{\partial w_q^{(L)}}{\partial \beta_{pq}^{(L)}} \cdot \frac{\partial \alpha_q^{(L)}}{\partial w_q^{(L)}} \cdot \frac{\partial C_0}{\partial \alpha_q^{(L)}} \tag{2.11}$$

These relative impacts of weights and activation's on all of the output neurons are the components that make up the gradient vector $\bigtriangledown C$ which are presented in Equations (2.12) and (2.13)

$$\frac{\partial C}{\partial \beta_{pq}^{(l)}} = \alpha_p^{(l-1)} f'(W_p^{(l)}) \frac{\partial C}{\partial \alpha_p^{(l)}} \tag{2.12}$$

$$\frac{\partial C}{\partial \alpha_p^{(l)}} = \sum_{p=0}^{P_{l+1}-1} \beta_{pq}^{(l+1)} \alpha_p^{(l-1)} f'(W_p^{(l+1)}) \frac{\partial C}{\partial \alpha_p^{(l+1)}} \tag{2.13}$$

The gradient vector, which contains all of the weights and biases in the NN, shows the magnitude of each component on the cost function. The number of hidden layers and the number of neurons in each of the layers is chosen based on experiments (i.e. grid search to optimise parameter settings).

Preferably, the activation of a neuron is bounded, e.g. to prevent erratic behaviour when receiving input unlike data present in the training set. Therefore a Sigmoid function (Equation (2.14)) is used to bring back the weighted sum to a value between 0 and 1. The Sigmoid activation function: α_σ, also known as the logistic curve, is commonly used in NNs.

$$\alpha_\sigma = \frac{1}{(1+e^{-W_p})} \tag{2.14}$$

A new technique is called the rectified linear unit (ReLU) (Xu, Wang, Chen, & Li, 2015) which found its origin in biology, where negative values get a value of 0 and the positive values keep their value. The rationale behind this is that a neuron cannot be more "inactive" than 0, while a neuron that is activated strongly has more priority than that of one that is barely activated.

The downside of using ReLU as an activation function is that a neuron can "die". A neuron is considered dead when its weights are stuck on negative values, thus always resulting in an output of 0. Therefore it no longer contributes to the error term used in training, is no longer updated, and is no longer used in the network. This potentially results in a substantial part of the NN being unused, but still using computation time (Volpi & Tuia, 2016), where the problem of dying neurons increases as the size of the network increases due to the vanishing gradient problem (Hochreiter, 1998).

There are variants to ReLU: Leaky ReLU, Parametric ReLU (Equation (2.15)) (Trottier, Gigu, Chaib-draa, et al., 2017), Exponential Linear ELU (Clevert, Unterthiner, & Hochreiter, 2015) (Equation (2.16)) and ReLU-6 (Equation (2.17)) (Krizhevsky & Hinton, 2010). These variants (except for ReLU-6) prevent dying neurons. In the Parametric ReLU and Exponential ELU a value for a must be determined, trough experiments.

$$\text{Leaky} : f(W_p) \begin{cases} 0.01W_p & \text{if } W_p < 0 \\ W_p & \text{if } W_p \geq 0 \end{cases}, \quad \text{Parametric} : f(W_p) \begin{cases} aW_p & \text{if } W_p < 0 \\ W_p & \text{if } W_p \geq 0 \end{cases} \tag{2.15}$$

$$\text{Exponential} : f(W_p) \begin{cases} a\left(e^{W_p} - 1\right) & \text{if } W_p < 0 \\ W_p & \text{if } W_p \geq 0 \end{cases} \tag{2.16}$$

$$\text{ReLU-6} : f(W_p) \begin{cases} 0 & \text{if } W_p < 0 \\ W_p & \text{if } W_p \geq 0 \text{ and } W_p < 6 \\ 6 & \text{if } W_p \geq 6 \end{cases} \tag{2.17}$$

The total number of connections between neurons (or nodes) in a network with 100 input nodes, 2 hidden layers each consisting of 20 nodes and a output layer of 10 nodes is then $100 \cdot 20 \cdot 20 \cdot 10 = 400.000$. These are only the weighted connections since every node, after the input layer also has a bias which is $20 + 20 + 10 = 50$ more bias connections, which means that there are 400.050 total connections in this NN.

2.3.3 Extreme Gradient Boosting (XGB)

Due to the rise and extensive documentation on Extreme Gradient Boosting with very promising results in other fields, this was also implemented and investigated (Chen & Guestrin, 2016). No other articles were found implementing XGB for tagging purposes in any language. XGB is an enhancement of Gradient Boosting (Chen & Guestrin, 2016). The enhancement lies in the fact that it is a more regularised model formalisation to control overfitting. Gradient boosting itself is a machine learning technique which can be used for regression and classification. Using decision trees it creates a prediction model from an ensemble of weak prediction models. The methods XGB uses are explained in more detail to understand the workings in the final model.

In a classification decision tree two errors can occur: bias- or variance-related errors. When there is a high bias and low variance the model underfits and when there is a low bias with high variance the model overfits. An accurate model has low bias and low variance as shown in Table 9.

	Low variance	High variance
High bias	Underfits	Wrong model
Low bias	Correct model	Overfits

Table 9. Bias and variance in XGB

A solution to prevent a model to overfit is bootstrap aggregation, also known as bagging (Breiman, 1996). In bagging the data in the training set is resampled with replacement.

The final model is the average of the models that were resampled. If the models overfit then the different models should overfit in different areas and therefore taking the average reduces the variance error.

Another method to solve overfitting is Random Forrest (RF) (Ho, 1995). RF has an additional component next to bagging. RF also randomly takes different columns (features) to be used in the model. Since RF adds an additional layer of randomness this reduces the overfitting even more than just bagging.

To prevent underfitting a method: Adaptive Boosting (AdaBoost) (Freund, Schapire, & Abe, 1999) can be applied. AdaBoost is a sequential method in contrary to bagging and RF which can be done in parallel. When a simple classifier is examined which needs to predict only two classes: a round or a triangle as shown in Figure 4, it works as follows.

After each round the weights of the wrong predicted observations are increased, while decreasing the weight of the correct observations.

This process is repeated until a predetermined number of rounds is passed. The errors in the training model in each round are summed to the final model.

Rather than simply putting a cutoff point on either X_1 or X_2, the AdaBoost made a more sophisticated and accurate model to reduce bias related errors.

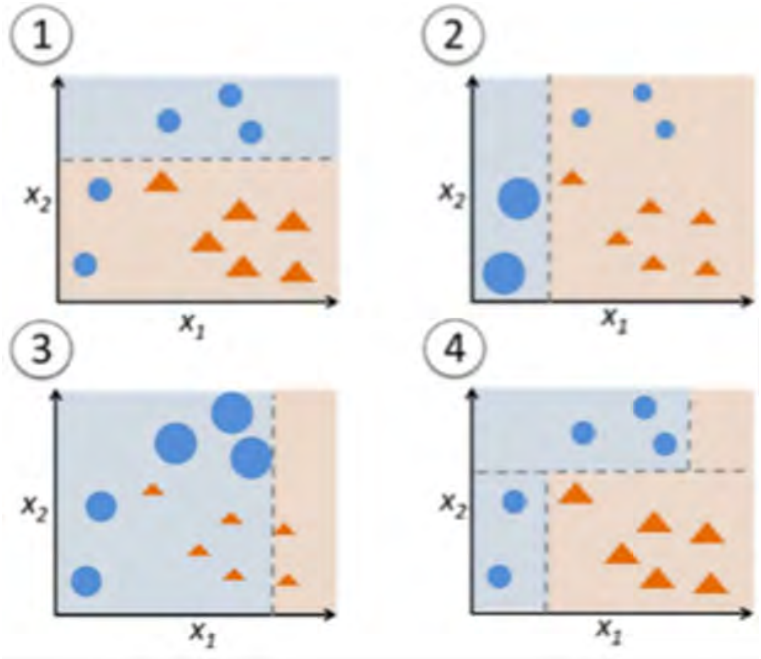

Figure 4. AdaBoost (Gautam, 2018)

Gradient Boosting (GB) is a more common technique to use then AdaBoost (Friedman, 2002).

The idea of GB is similar to AdaBoost but it works with regression techniques. An initial classifier is fitted on the training data and the errors are observed. In every round of GB the errors are compensated. Therefore instead of learning the initial concept the learning takes place on the errors.

From the start there are initial residuals, then the model is fit. Afterwards the whole model is taken, with recalculations of the errors and fitted to the residuals again. This process is repeated until a certain threshold is met.

The residuals are the gradient of the Root Mean Squared Error function (RMSE) (Gilroy, Hirsch, & Cohn, 1990). The problem with classification problems is the fact that there are no residuals, which necessitates a different loss function. An example is the loss function from the logistic regression: logistic loss.

The gradient of logistic loss are called pseudo-residuals (Friedman, 2002). However the

method of building the final model remains the same, but instead of fitting on the residuals one fit on the pseudo-residuals.

Since XGB can be used as a classifier, it can also be used to predict the tag of a word, based on features. Scikit-learn (Pedregosa et al., 2011) also made XGB available for regression and classification problems, with parameters that can be changed. The first parameter is the *loss* which can be set to either *deviance* or *exponential*. This parameter determines which loss function is to be optimised. When the *deviance* option is chosen the same loss function is used as in logistic regression with probabilistic outputs. In case of exponential the AdaBoost algorithm is used, whereof the workings has previously been described.

The second parameter that can be adjusted is the *learning rate*. Since the newly build trees correct the residual errors, obtained from the previous trees, there is a possibility that the model fit rather quickly. This results in overfitting of the training dataset, which has to be avoided. To overcome this problem one can use a weighting factor on the corrections, that the new constructed tree adds to the model. The weighting factor is also know as the learning rate. The learning rate is a float where larger values, result in less corrections for each newly added tree. When the learning rate is set relatively low, more trees are needed to find a good solution since the correction for each tree on the model is smaller.

The number of trees that are build is captured my the *n_estimators* parameter. There is a trade off between computation time and performance, but in general it is safer to set the number of estimators high. This is especially the case when the learning rate is set low, as previously explained.

The *max_depth* parameter determines the maximum number of splits a tree is allowed to make, before a new tree is build.

2.4 Sequential Models

Sequential models can be used in various ways to help stem or tag a word not only based on prefixes and suffixes, but more importantly on the role of the word in a sentence. There are many ambiguous words that are written the same but do not have the same meaning. An example is *Ik wilde de vrouw helpen* (I wanted to help the woman) and *de wilde vrouw hielp mij* (the wild woman helped me). When only the word *wilde* is examined it can either be a verb (to want) or an adjective (wild). Many promising results have been observed using Long-Term-Short-Memory (LSTM) models (Chung, Gulcehre, Cho, & Bengio, 2014) for natural language processes.

Another method is called Pervasive Attention which uses a 2D Convolutional Neural Network for Sequence-to-Sequence Prediction (Elbayad, Besacier, & Verbeek, 2018).

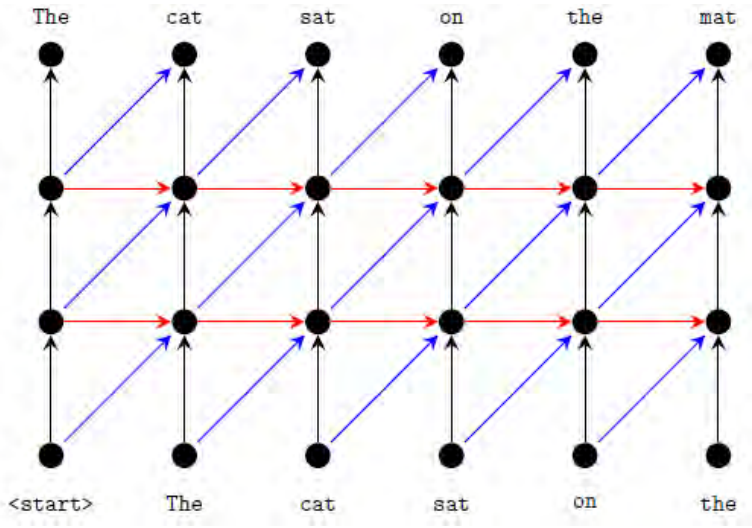

Figure 5. Pervasive attention representation (Elbayad, Besacier, & Verbeek, 2018)

Figure 5 is a graphical representation of a decoder network topology with two hidden layers. Nodes at bottom and top represent input and output respectively. Horizontal connections are used for Recurrent Neural Networks (RNN), diagonal connections for Convolutional Neural Networks (CNN). Vertical connections are used in both cases. Parameters are shared across time-steps (horizontally), but not across layers (vertically).

In this case it is used to predict what the next word is going to be, but can also be used to determine the next possible tag. A problem that possibly occurs in using sequential models on business or social media texts is that not always full sentences are used. When a table of contents is included it is hard for a sequential model to determine what a word is. Social media texts are most often unstructured and lack grammar spelling. These can create noise and decreases accuracy of the stemming algorithms.

Another problem with a pure sequential model is finding the first word and determining the tag. This would ideally be a combination of examining the suffixes and prefixes of a word. In case the first tag is incorrectly predicted the error will likely propagate to subsequent tags.

3 METHODS

The scope of this thesis limits itself to the Dutch language, and only evaluating three tagging algorithms: *MLR*, *NN*, *XGB*, due to time restrictions and data availability. For the actual stemming a new rule based stemmer is developed, part of the Bag & Tag'em (BT) algorithm.

Frog is generally regarded as highly accurate for the Dutch language and may be considered a benchmark representing the state of the art in terms of performance. Frog determines both tag and stem (or lemma) based on a single word rather information from the sentence that word was taken from. Since PoS-Tagging shows promising results in literature and due to the availability of the Frog algorithm as a ground truth, we investigated the feasibility of a novel approach that combines a token-based (single-word) tagging module and a rule-based stemming module.

The stemming module uses the tag from the tagging module to determine which rules to apply to reduce the word to its stem. Sequential models were out of scope for the research described in this thesis, as we were not aware of any available data sets that could be a ground truth. The following sections will describe how the final product was designed and constructed, with next steps such as extension by sequential models discussed in Section 5.3. A visual representation of the three modules used in this thesis are presented in Figure 6.

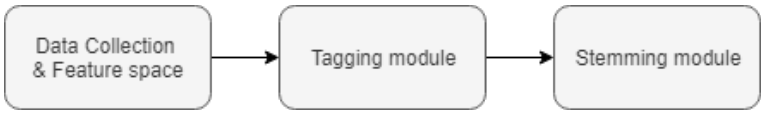

Figure 6. Flowchart of the three modules in the BT algorithm

3.1 Data Collection & Feature space

3.1.1 Data Collection

To determine which algorithms to use in the final product, documents were collected to conduct analysis and for training purposes. To increase the probability of "correct" usage in the Dutch language certain types of publications are preferred over others. An example can be Social Media information, where sentences can contain many grammar mistakes. The probability of correctly constructed Dutch sentences is likely higher in legal documents. A web crawler tends to "grab" all information about a certain topic and does not discriminate necessarily on the quality of grammar, which is another downside of using web crawlers.

This was done by downloading publicly available collective labour agreements (CLA) as PDF files from the website of the largest Dutch trade union: FNV. Besides the CLA's, pages from Wikipedia and Dutch children's books were used to diversify the words used. The PDF documents where parsed using a combination of Tika parser (Mattmann & Zitting, 2011) and Tesseract (Smith, 2007), both of which are open source in Python 3.6. All programming was conducted in the Anaconda environment using Spyder (3.3.4) and Jupyter Notebooks (5.7.8).

In total 252 CLA's, five pages from Wikipedia and three children book chapters were parsed. The Wikipedia pages contained information on: Amsterdam, The Dutch Royal family, Koala's, the flooding of the Netherlands in 1953 and famous Dutch painter: Rembrandt van Rijn. The three children book chapters were: *Spijt* from Carry Slee, *Pluk van de Petteflet* by Annie MG Schmidt and *Hoe overleef ik de brugklas?* by Francine Oomen.

The parsed documents were tokenised using the same heuristics as previously described in Section 2.1.2. Although it is common practice to perform stop word removal as described in Section 2.1.3 it was not used for our data collection purposes. The rationale was that this would remove information for the training of the different models.

The unique words were converted into a Pandas dataframe for further analysis. To prevent overfitting on words due to frequency of occurrence, only the unique words were added. If the word *de* (the) would be contained in the dataset many times, it could result that the

feature *de* is more prone to label words as a BW (adverb), instead of a ovt (simple past tense). The dataframe was connected with the Frog algorithm using LaMachine (van Gompel & Hendrickx, 2019) and a virtual machine, to obtain the tag for each word.

Since Frog uses a taxonomy of tags that is more complex than needed for our classification purposes, Frog tags were converted into six new tag categories, further explained in Table 10.

Because in the Dutch language the verbs can be contained in these three categories only these were used for simplification purposes. There is a difference in how a sentence is constructed but looking at the prefix and suffix of the main verb are these sufficient, as explained in Section 1.2.

Tag	Translation	Example	Dutch
ZNW	Noun	Dog	Hond
BVNW	Adjective	Big	Groot
BW	Adverb	Other	Ander
ott	Present tense	Walk	Loop
ovt	Simple past tense	Walked	Liep
vtt	Past perfect tense	Walking	Gelopen

Table 10. Tags explained

The final dataset consisted of 25.389 unique words after cleaning, with the distribution of the tags shown in Figure 7. An additional dataframe was built combining the three verb categories into a single verb category (WW) for experimental purposes. During writing of the code all words that were not a verb, noun or adjective were labelled BW. This contradicts the information from Section 1.2, since there are a lot more categories. For stemming purposes it is more convenient to list these as a BW due to the fact that these words will not be stemmed regardless.

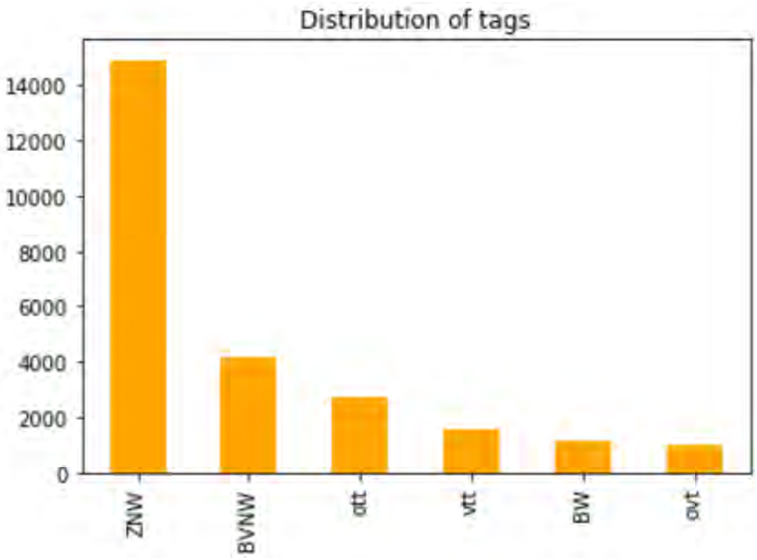

Figure 7. Distribution of tags in the dataset

3.1.2 Feature space

The words were converted into a feature space word. A feature space word is a word with an _ symbol added to the beginning and end of the word. For example the word *walking* would be converted into *_walking_*. This ensures that in the next step it is still clear what the first and final letters of the word were.

Since algorithms generally cannot handle strings but need vectors to be able to conduct analyses on, a new vector space was created. To construct the dependent variables in the various algorithms, one can use N-grams of characters in words (Cavnar, Trenkle, et al., 1994). This means that single letters are matched with their N neighbours as an input feature of a word. An example is shown in Table 11 where the N-grams are converted into N-gram vectors in a later stage to be used in the algorithms.

This is slightly different from the technique described in Section 2.3, where only the last three letters were examined. Using N-grams other potential landmark features are taken into account, including the beginning and middle parts of a word.

Word	1-gram	2-gram
ren	r,e,n	re,en
loop	l,o,o,p	lo,oo,op

Table 11. N-gram example

The feature space consisted of $28 \cdot 26 = 728$ different vectors. There are 26 letters in the Dutch alphabet, together with the new _ symbols on the beginning and the end of the word. The word *ren* would have a value of one in the vectors: _r, re, en, n_ and a zero in all other vectors. This is done using an encoder provided by (Pedregosa et al., 2011) which uses one hot coding.

The distribution of the features was skewed, which is to be expected in textual data. Some combinations are more common (e.g. *en*) than others, while some do not occur at all (e.g. *xq*) in the Dutch language. Based on experiments and manual inspection, features that occurred less than 200 times in the data were removed, to decrease possible noise in the algorithms later on. In total there are four different datasets with the same words but differences in tag construction and if feature selection was conducted. An overview is given in Table 12.

Dataset	Verb tags grouped	Feature selection
1	No	No
2	No	Yes
3	Yes	No
4	Yes	Yes

Table 12. Attributes of datasets

Since the distribution of the different tags (Figure 7) was imbalanced the different train and test sets were carefully constructed. First a stratified (80/20) sample was obtained from dataset number one. Stratifying is a technique that does not randomly take 80% of the data but 80% of each of the categories. To ensure that all datasets train on the same words, the same words were used from the initial train and test set.

If the dataset is not balanced it could lead to biased predictions and thus a misleading accuracy. An example of this can be found in a fraud detection system. If 1% of the transactions are fraudulent, the model can get an accuracy of 99% by classifying all transactions as non-fraudulent transactions. The model is too biased towards non-fraudulent transactions.

There are multiple ways to solve imbalance in a dataset. Two of these are Random Under-Sampling and Random Over-Sampling. Random Under-Sampling randomly removes observations of the majority class. This helps balancing the dataset but the removed observations could have contained valuable information, dropping of which could lead to a bias or otherwise hinder performance (Drummond, Holte, et al., 2003).

Random Over-Sampling is a technique where observations of minority classes are duplicated randomly. This helps balancing while ensuring that there is no loss of information. The downside to this technique is that it is prone to overfitting on the data since the same information is duplicated and therefore potentially weighted disproportionately (Drummond, Holte, et al., 2003).

In order to keep information while mitigating the risks of over-sampling, we used an approach called SMOTE (Chawla, Bowyer, Hall, & Kegelmeyer, 2002). SMOTE is an over-sampling method in which minority classes are over-sampled by creating synthetic observations. This method consists of four steps where Figure 8 shows the end result:

1. SMOTE identifies the feature vector and its nearest neighbour.
2. The linear distance is taken between the two feature vectors.
3. On the line created at step 2, a new synthetic observations is placed based on a random number ranging from 0 to 1. If the random number is 0.5 then the new synthetic observation is placed right in the middle.
4. The previous steps are repeated until a predetermined number of new observations are added. An important note to this is that only the original observations are used to create new synthetic observations.

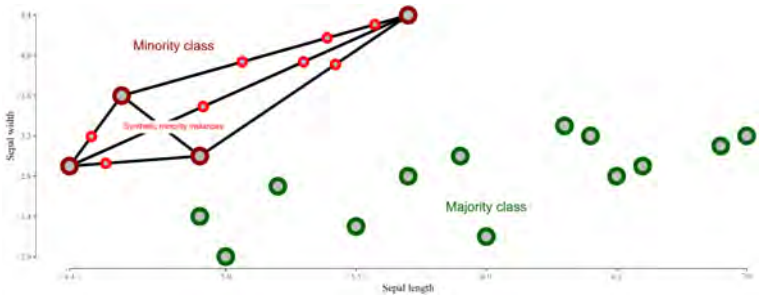

Figure 8. Workings of SMOTE obtained from (Rikunert, 2017).

A visual representation on the various steps from Section 3.1, is presented in Figure 9. Section 3.2 will continue on the manner how a tagging algorithm is build using the train and test set.

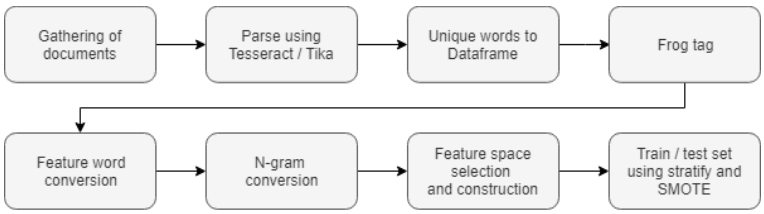

Figure 9. Data collection & Feature space flowchart

3.2 Tagging algorithms

Three different algorithms were investigated to see which of these could predict the correct tag in the test set. The three algorithms are: MLR, XGB, NN. To evaluate the accuracy of these algorithms a performance measuring score was used. In classification algorithms generally the precision and recall are used to determine the performance. To calculate the precision and recall Table 13 is used. For simplification purposes only nouns and non-nouns are explained. If the actual data states that a word is a noun and so does the algorithm then this is a true positive (TP). If the actual data states that a word is a noun but the algorithm predicts a non-noun then this is a false negative (FN). If the actual data states that a word is a non-noun but the algorithm predicts a noun then this is a false positive (FP) (Olson & Delen, 2008). The equations for the precision and recall of a class k, are shown in Equation (3.1).

	Actual	
Predicted	True positive (TP)	False positive (FP)
	False negative (FN)	True negative (TN)

Table 13. Performance measuring

$$\text{Precision}_k = \frac{\text{TP}_k}{\text{TP}_k + \text{FP}_k}\ , \text{Recall}_k = \frac{\text{TP}_k}{\text{TP}_k + \text{FN}_k} \tag{3.1}$$

The actual used performance measure is the F_1-score which is the harmonic mean of precision and recall (Chinchor, 1992). This is the most common performance measure in information retrieval, despite being criticised (Hand & Christen, 2018). The article states that precision and recall are equally important for the F_1-score which is not necessarily the case. While this is a valid criticism, the regular F_1-score is used as shown in Equation (3.2) for class k.

$$F_{1_k} = 2 \cdot \frac{\text{Precision}_k \cdot \text{Recall}_k}{\text{Precision}_k + \text{Recall}_k} \tag{3.2}$$

The algorithms were implemented using packages from Scikit-learn (Pedregosa et al., 2011). To improve the predictions of the algorithms parameters were tuned, as previously described in Sections 2.3.1, 2.3.3 and 2.3.2. The tuning was performed trough grid searches. A grid search takes a range of values for pre-selected parameters and calculates the model's performance (i.e. the F_1-score) attained with the different combinations of parameter values, to finally return the optimal parameter settings.

Since the datasets are not entirely the same, all algorithms were grid searched for each dataset separately, but with the same range in a grid search. This resulted in the following hyperparameters presented in Table 14. The hidden layer size of the optimal parameters per dataset

are tuples. This means that in (α,β): α represents the number of hidden nodes in the first hidden layer, and β the number of hidden nodes in the second hidden layer.

Algorithm	Parameter	Range	1	2	3	4
MLR	Penalty	All	L2	L2	L2	L2
MLR	Solver	All	Saga	Saga	Saga	Saga
MLR	Max iterations	45-75	68	65	61	58
NN	Hidden layers	1-5	2	2	2	2
NN	Hidden layer size	20, 50, 100	(50,20)	(50,20)	(20,20)	(20,20)
NN	Activation	All	ReLU	ReLU	ReLU	ReLU
NN	Solver	All	Adam	Adam	Adam	Adam
NN	Max iterations	50, 100, 200, 1000, 3000	3000	3000	3000	3000
XGB	Maximum depth	10-20	18	17	15	14
XGB	Learning rate	0.01, 0.05, 0.1, 0.15	0.1	0.1	0.1	0.1
XGB	Number of estimators	50, 100, 300, 500, 1000	300	300	300	300

Table 14. Algorithm tuned parameters and optimal parameters based on dataset

Table 14 shows no big differences in optimal parameters among the various datasets. The only variation in optimal parameters are: The maximum number of iterations for the *MLR*, the hidden layer sizes from the *NN* and maximum depth from the *XGB*. The value for the previously mentioned parameters decreases as the number of feature vectors and possible tags decreases.

The results of the datasets using the optimised parameters are shown in Tables: 15, 16, 17, 18. The micro average F_1-score is defined as the sum of all correctly predicted observations divided by the total number of observations, as shown in Equation (3.3).

$$\text{Micro average} = \frac{\sum_{i=1}^{k} \text{TP}_i}{\sum_{i=1}^{k} \text{TP}_i + \sum_{i=1}^{k} \text{FP}_i} \tag{3.3}$$

The macro average F_1-score is defined as sum of the precision on each of the classes divided by the total number of classes K, as shown in Equation (3.4).

$$\text{Macro average} = \frac{\left(\frac{\sum_{i=1}^{k} \text{TP}_i}{\sum_{i=1}^{k} \text{TP}_i + \sum_{i=1}^{k} \text{FP}_i}\right)}{K} \tag{3.4}$$

The weighted average F_1-score is defined as the correct predicted observations multiplied by the support and divided by the number of observations as shown in Equation (3.5).

$$\text{Weighted average} = \frac{\sum_{i=1}^{k} \text{TP}_i \cdot \text{Support}_i}{\sum_{i=1}^{k} \text{Support}_i} \tag{3.5}$$

The support is the number of observations per tag in the test set. On all the datasets the XGB-Classifier outperforms the other algorithms, based on the weighted average.

Dataset 1	Neural Network	XGB	MLR	Support
BVNW	0.59	0.66	0.60	824
BW	0.28	0.27	0.13	219
ZNW	0.79	0.83	0.81	2982
ott	0.54	0.61	0.57	542
ovt	0.16	0.18	0.18	201
vtt	0.61	0.69	0.65	310
micro avg	0.68	0.74	0.71	5078
macro avg	0.49	0.54	0.49	5078
weighted avg	0.67	**0.72**	0.68	5078

Table 15. F_1-scores on the first dataset

Dataset 2	Neural Network	XGB	MLR	Support
BVNW	0.59	0.67	0.58	824
BW	0.25	0.28	0.18	219
ZNW	0.78	0.84	0.81	2982
ott	0.57	0.60	0.54	542
ovt	0.16	0.19	0.15	201
vtt	0.65	0.68	0.65	310
micro avg	0.66	0.75	0.71	5078
macro avg	0.50	0.54	0.48	5078
weighted avg	0.67	**0.73**	0.68	5078

Table 16. F_1-scores on the second dataset

Dataset 3	Neural Network	XGB	MLR	Support
BVNW	0.57	0.67	0.60	824
BW	0.21	0.26	0.17	219
WW	0.59	0.66	0.59	1053
ZNW	0.78	0.83	0.80	2982
micro avg	0.68	0.76	0.71	5078
macro avg	0.53	0.61	0.54	5078
weighted avg	0.68	**0.74**	0.69	5078

Table 17. F_1-scores on the third dataset

Dataset 4	Neural Network	XGB	MLR	Support
BVNW	0.59	0.67	0.58	824
BW	0.27	0.40	0.17	219
WW	0.61	0.66	0.59	1053
ZNW	0.80	0.83	0.80	2982
micro avg	0.71	0.76	0.71	5078
macro avg	0.57	0.63	0.53	5078
weighted avg	0.74	**0.75**	0.70	5078

Table 18. F_1-scores on the fourth dataset

The best results were obtained from dataset four. The problem is that in limiting the number of possible tags it should improve the accuracy any way. Since it is necessary to give a more accurate tag than just WW (verb) to the stemming algorithm an additional XGB-classifier model was trained. This takes as input the WW converted back to the three original verb tags: ott, ovt, vtt together with the incorrectly classified verbs.

The latter are the words that the model incorrectly classified as a verb (i.e. false positives), which were isolated so as to retain potential information leading to the initial classification error. These words were renamed to BW2, but discarded afterwards.

A visual representation of the various steps explained in this Section 3.2, is presented in Figure 10.

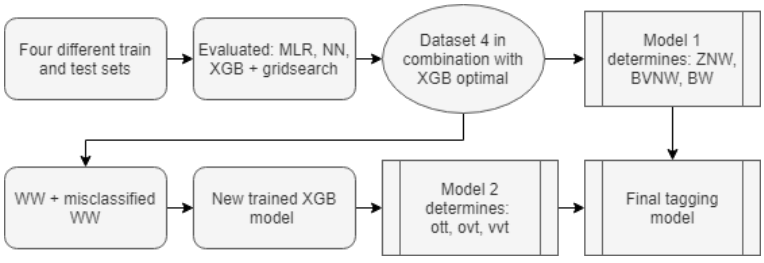

Figure 10. Tagging module flowchart

3.3 Stemming algorithms

The quality of a stemmer is determined by its ability to accurately reduce words belonging together to a common stem while ensuring that words that do not belong together yield different stems (Moral, de Antonio, Imbert, & Ramırez, 2014). There are two main errors that determine the precision and both relate to the aggressiveness of the stemmer in removing parts of the words. If the stemmer is not aggressive enough, insufficient prefixes and/or suffixes are deleted which can result in the incorrect stem. This type of error is called under-stemming, since the stemmer stripping of words is under the required level.

The other type or error is over-stemming, where the stemmer is too aggressive and removes too many parts of the word. This leads to clearing parts of the morphological root, which results in loss of semantic information and falsely reducing words to a common root. (Porter, 2001) makes two distinctions in over-stemming. The actual over-stemming where the deleted suffix results in a change of the meaning of the stem (the deleted part is a suffix but belongs to the root). The other distinction is so called mis-stemming. In this case the deleted part is not actually a suffix but part of the root. An example of mis-stemming was also given in (Porter, 2001): when a rule was implemented for removing the suffix *-ly* it goes well for the word "cheaply", since this results in "cheap". The problem occurs when this is also done for the word "reply". Following the same rule "reply" stems to "rep", which is normally used as an abbreviation for the semantically different word "representative".

Under- and over-stemming decrease the performance of the stemmer and therefore the quality. There are some solutions to decrease these errors. One solution to prevent over-stemming is to set a minimum length of the stem. This decreases the probability that a suffix is removed even though this was not a suffix to begin with. For example: removing the suffix *-en* from the word "ren" (run or a rabbit cage) would result in a "r". Typically the stem needs a minimum of two or three letters. Removing suffixes from words is very tricky due to the differences in languages and exceptions that occur. Carefully constructed suffix elimination rules are therefore critical.

A theoretical example of over and understemming is as follows: when we look at Table 19. Frog determines that there are four different stems in the total of six words: *walk*, *walking*,

car and *caretaker*.

The stemmer has has two different stems: (*walki* and *walke*), for the ground truth stem of Frog of *walking*. This means that the stemmer is not aggressive enough, since according to the stemmer the words *walking* and *walked* are not the same even tough Frog says it is. This concept is called understemming.

The stemmer has only one stem *car* for two Frog stems: *car* and *caretaker*. The stemmer was too aggressive in this case and the difference between a *car* and *caretaker* is no longer visible. This concept is called overstemming.

A way in calculating the severity of understemming and overstemming and how it is applied in the thesis, is described in Section 3.4.

Word	Frog	Stemmer
walk	walk	walk
walking	walking	walki
walked	walking	walke
car	car	car
cars	car	car
caretakers	caretaker	car

Table 19. Example of under and overstemming

The BT algorithm divides the dataframe into five subcategories which were obtained from the tagging module. Each subcategories represents a tag which follows different stemming rules.

3.3.1 Verbs (WW)

The first step of stemming a verb is to determine if the verb is a compounded verb. This is done by comparing the first three characters of the verb and a list of known compounded prefixes. Known compounded prefixes are: *af*, *bij*, *in*, *op*, *over*, *uit*. An example (Table 20) is given why it is important to know if the word is compounded especially in the past

perfect tense. All the verbs has its root from the verb *lopen* but differ in meaning. If only prefixes and suffixes were to be removed the present simple form of the word *ingelopen*, which is *inlopen*, would not be stemmed to the same stem. Therefore the words are split in the beginning if the verb starts with any known compounded prefixes.

After testing there were a few verbs that were not stemmed properly due to this way of splitting. An example was the verb *opgeven* (give up), the actual stem should be *opgev* since the past perfect tense is *opgegeven* (given up). These verbs were added to an exception list which will be discussed later on in Section 3.3.5.

Verb (vtt)	Translation	Stem
gelopen	walked	lop
afgelopen	finished	aflop
ingelopen	walked in	inlop
opgelopen	sustained	oplop
overgelopen	defected	overlop
uitgelopen	ran out	uitlop

Table 20. Example compounded verbs

The Dutch language contains roughly 235 irregular verbs (Haeseryn, Romijn, Geerts, Rooij, & Van den Toorn, 1997). These verbs have different vowels and structures in different tenses, although they should stem to the same root. Again the example of the irregular verb *lopen* (present simple) shows us that the past tense is *liepen* (past simple). These verbs follow some rules of the same rules as a regular verb but the stemmed version of the irregular verbs are added into an exception list.

For stemming it is important to know if a verb is singular or plural. Therefore an additional column is added to the dataframe indicating if the verb is singular or plural. Verbs that have characteristics of being an infinitive (suffix *-en*), obtain plural and all others singular. There are infinitives where the suffix is *-enen* for example *rekenen* (to calculate), *zegenen* (to bless). The singular form of these verbs also have suffix of *-en*, to prevent that the singular form is processed as an plural these verbs are added to the exception list.

For readability purposes and stemming of adjectives the stem that is used as the actual stem is the first-person singular present simple. Therefore additional rules are implemented to transform the stem to the first-person singular present simple form. The affix stripping rules are shown in Table 21. The rules are applied in a particular order based on Dutch grammar rules, as explained in Section 1.2.

	Condition 1	Condition 2	Condition 3	Rule
1	time = vtt	starts with: ge		-ge
2	time = vtt	ends with: d		-d
3	Length >3	ends with: en		-en
4	time = ovt	Length>3	ends with: dt	-t
5	ends with: t	second to last letter is: d		-t
6	ends with: t	is regular	second to last letter is not a vowel	-t
7	ends with v			v = f
8	ends with z			z = s

Table 21. Verb stemming rules

In the Dutch language no word either ends with a *v* or a *z*. There are exceptions but these words are "borrowed" from other languages. Borrowed words for example are quiz and jazz and are likely to be stemmed incorrectly. Due to the fact that the number of words that are borrowed is limited a few of these are also included in the exception list. After removing the suffixes of a verb thus reducing the verb to its root, the last letter is examined. If the last letter is a *v*, then it is replaced by the letter *f*. If the last letter is a *z*, then it is replaced by the letter *s* (rule 7 and 8) and are know in the Dutch language as the *valse s en valse f* (foul s and foul f).

An example on how verbs are stemmed following the rules from Table 21 is shown below in Table 22 and worked out in Table 23.

Verb	Translation	Time	Regular	Final stem
gelopen	walked	vtt	yes	loop
ren	run	ott	yes	ren
gaven	gave	ovt	no	geef
wordt	become	ott	yes	word

Table 22. Example of verbs in stemming algorithm

	1	2	3	4	5	6	7	8
gelopen	lopen	lopen	lop	lop	lop	lop	lop	lop
ren	ren	ren	ren	ren	ren	ren	ren	ren
gaven	gaven	gaven	gav	gav	gav	gav	gaf	gaf
wordt	wordt	wordt	wordt	wordt	word	word	word	word

Table 23. Stemming rules applied to the root for verbs

To check if a verb is irregular the exception list is called after rule 5 to decrease the size of the exception list as much as possible, due to the fact that not all forms of a verb need to be present. The word of Table 23 is not the actual final stem since other rules need to be applied, this is the so called root.

When the verb is stemmed to the root, doubling of vowels is needed to bring it back to the actual stem. The first-person singular present simple form of *lopen* is *loop*. Following the stemming rules as previously described the root of the word *lopen* becomes *lop*. Therefore an additional *o* must be placed before the last letter.

This is also the case for the letters *a*, *e* and *u*. To see if a vowel is doubled the algorithm looks at the last three letters of the stemmed word, which was originally a plural. If the first letter is a non-vowel and the second letter is a vowel, then the second letter gets doubled. To go back to the previous example of *lopen* which was determined to be a plural verb in the first step, following the stemming rules suffix *-en* is removed. Afterwards the last three letters *lop* (in this case the full word) is examined. *L* is the first letter and a non-vowel, *o* is the second letter and therefore is doubled resulting in *loop*, which is the first-person singular

present simple of *lopen*.

The final step in ensuring the correct stem, is removing double letters at the end of a verb in case this is a non-vowel. This is due to the fact that these do not exist in the Dutch language.

3.3.2 Adjectives (BVNW)

The adjectives require less rules to bring the adjective back to the root. The root of an adjective is defined as its shortest possible form, e.g. stripping suffixes indicating superlatives or grammatical gender. As a design choice superlatives were brought back to the normal form. For example *goed* (good), *beter* (better), *beste* (best) were all stemmed back to *goed* trough the exception list. First the following suffixes are removed in a particular order, as shown in Table 24. The adjectives can only be stemmed if the length after removing a suffix is still greater than two. An example of the workings is shown in Table 25.

	Condition: Adjective ends with	Rule: Remove
1	-er	-er
2	-ste	-ste
3	-ende	-ende
4	-en	-en
5	-e	-e

Table 24. Adjectives stemming rules

Adjective	Translation	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Irregular check	Actual Stem
mooi	beautiful	mooi	mooi	mooi	mooi	mooi	mooi	mooi
mooiste	most beautiful	mooiste	mooi	mooi	mooi	mooi	mooi	mooi
goed	good	goed	goed	goed	goed	goed	goed	goed
beter	better	bet	bet	bet	bet	bet	goed	goed
beste	the best	beste	beste	beste	beste	bet	goed	goed

Table 25. Stemming rules applied to adjectives

The irregular check looks for the word *bet* in the exception list and not for *beter* and *beste*. Again this was done to reduce the size of the exception list similar to the verbs. After the suffixes are removed the adjective is handled in the same way as the verbs, as described in the previous Section 3.3.1.

3.3.3 Noun (ZNW)

The stem of a noun is defined as the singular form of the noun. Therefore the first step is to determine if a noun is plural or singular. If the noun ends on suffix *en*, *jes* or *s* it is considered to be plural. There is an additional requirement on the suffix *s* to be a plural and that is a reversed doubling of vowel method. Since a noun that is actually in singular form but ends on an *s*, it will probably have a double vowel before the *s*. Since this is not possible in plural nouns, the noun is not marked as a plural but as a singular noun. For example the noun *moeders* (moms) will be stripped of the suffix *s* but *vaas* (vase) will not be stripped from the *s* due to the double *aa*.

The plural nouns are then stripped of their respective suffixes, that marked them as a plural.

After the suffixes are removed the adjective is handled in the same way as the verbs, as described in the previous Section 3.3.1.

3.3.4 Adverb (BW)

Since the adverbs are words that are likely to be removed after stemming due to no information gain, are these also not stemmed. Names of countries, people or companies follow to many different rules to accurately stem them.

3.3.5 Exception list

As previously mentioned, it is not possible to stem all words due to irregularities. To reduce the size of this list, words were stemmed as much as possible and then added to an exception list.

The irregular verbs were obtained from a Dutch dictionary website (Mijnwoordenboek.nl, 2004). This website has a large database of almost all verbs, nouns and adjectives with their various forms.

There are four reasons why a verb was added to the exception list.

1. The present simple form already started with the prefix *ge*. Normally the prefix *ge* would classify the verb as a vvt, but the verb *geven* (to give) is in the present simple even tough it has a prefix *ge*.
2. In case the suffix is *enen* of an infinitive than the present simple singular form also has a suffix of *en*. This suffix would be stripped since the algorithm assumes this is an infinitive. An example is the verb *rekenen* (to calculate), the present simple singular form is *reken* (I calculate). When the standard rules are applied the algorithm would strip *reken* to *rek* (I stretch). To overcome this hurdle were these verbs also added.
3. Verbs that did not originate from the Dutch language were added. These are "borrowed" from other languages and therefore do not follow Dutch grammar rules. For example *kiten* (kite surfing), following the standard grammar rules the present simple singular form should be *kit* but it is written as *kite*.
4. Finally the irregular verbs, where the past tense singular form does not convert to the present simple singular form after affix stripping. The example that has been mentioned numerous times trough out this thesis is *lopen* (to walk). The present simple singular form is *loop* but the past simple singular form is *liep*. The algorithms needs

to "know" that this is the past tense and this is achieved by the exception list.

Adjectives that were added to the exception list were the superlatives that did not convert back to the shortest possible form after affix stripping. An example is *goed* (good), *beter* (better), *beste* (best) were all stemmed back to *goed* by design choice.

The benefit of this exception list is that it is relatively easy to add words that are stemmed incorrectly due to language exceptions, even tough the stemming module followed all the general rules. An overview of the stemming procedure is presented in Figure 11.

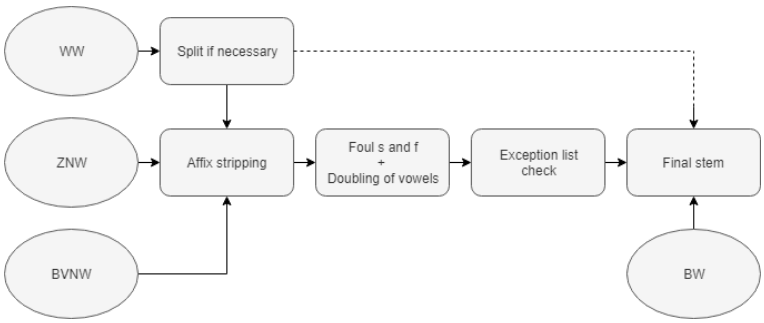

Figure 11. Stemming of the various tags flowchart

3.4 Model evaluation

To determine how accurate the entire model is a new evaluation dataset is used. This dataset is manually tagged and stemmed by the author of the thesis. The documents that were used were not previously seen by the tagging algorithm. Since the type of language that the models was trained on mainly focused on CLA's, it is interesting to see how the models perform on other types of text.

The description of the four documents are presented below and the actual texts are presented in Appendix E:

1. A chapter of the Dutch translation of Roald Dahl's book "the BFG" (i.e. "de GVR"), to see how the model handles non-existing words.
2. A newspaper article (Marcelis, 2019) on the earthquakes happening in the northern part of the Netherlands due to gas extraction. This article contains semi-business type language.

3. A Wikipedia page containing information on the University where this model has been developed (Vrije Universiteit Amsterdam) (Wikipedia, 2019).
4. A vacancy for a position at the (personal opinion of author) best football club in the world: Ajax. To examine how the model performs on business language text documents (Ajax, 2019).

The stemming algorithms are tested along three dimensions: computational speed, understemming and overstemming. The combinations and results of the various models will be discussed in Section 4.

The evaluated documents were stemmed and tagged manually by the author and was controlled by two other people. One was a fellow student who was double-checked by a primary school Dutch teacher. The Cohen's kappa was calculated to determine the accuracy of the manually checked file (Berry & Mielke Jr, 1988). The Cohen's kappa was calculated at 0.96% which is extremely high. Cohen's kappa is used to determine the inter-rater agreement on the document.

The words with different outcomes were analysed and changed where necessary. Some words contained typos and there was occasional disagreement about a tag, based on only the word. The documents were manually processed and Frog was not used to determine the tags taken as ground truth, so that evaluation of the performance of the Frog algorithm was possible.

An overview of the evaluation documents are given in Table 26. The distribution of tags shows that the ovt and vvt categories are underrepresented in this dataset. The number of words that were unknown to the CELEX database, will impact the performance of the Frog algorithm since Frog uses CELEX to determine the tag. If Frog is unable to determine the tag trough CELEX it uses an additional algorithm but the workings of this additional algorithm are unknown.

Dataset	1	2	3	4	Total
Words	390	340	443	340	1513
Unique Words	203	177	230	182	670
CELEX unknown	27	17	40	19	102
BW	75	72	65	69	187
ZNW	48	47	88	47	224
BVNW	33	17	33	31	110
ott	34	24	41	18	103
ovt	10	8	0	10	24
vtt	3	9	3	7	22

Table 26. Evaluation datasets overview in absolute numbers

To calculate understemming the unique stems of the manual checked file were taken as the ground truth. The reason for using the manual checked file and not the Frog algorithm, was due to the fact that also the Frog algorithm made errors in tagging. The unique stems were obtained using the various algorithms. The number of unique stems produced were divided by the number of unique manual stems as shown in Equation (3.6). This gives an accuracy percentage to compare the various algorithms.

$$\text{Understemming} = \frac{\text{Number of unique manual stems}}{\text{Number of algorithm stems}} \tag{3.6}$$

To calculate overstemming the unique stems of the specific algorithm was taken as the ground truth. The unique stems were compared to the number of unique manual stems as shown in Equation (3.7). This gives an overstemming accuracy percentage to compare the various algorithms.

$$\text{Overstemming} = 1 - \frac{\text{Number of algorithm stems}}{\text{Number of unique manual stems}} \tag{3.7}$$

Computational speed was determined by using cell magic in Jupyter Notebook. The averages were taken from 100 test runs and 100 loops each.

4 RESULTS

The evaluation datasets were processed to determine understemming, overstemming and the computational speed of the various stemming algorithms. These results are discussed in Section 4.1. The *Total* that is mentioned in these sections is not the sum of the various documents, but the four databases concatenated. This is done to ensure that the number of words in a document does not influence the average on multiple documents.

4.1 Accuracy of the models

Table 27 shows that the tagging algorithms outperform, looking at understemming (the algorithm does not stem aggressively enough), the non-tagging algorithms on every document. This is to be expected since the stemming algorithm has more information on how to accurately stem.

There is some variation in the understemming accuracy among the various documents, but all tend to a follow trend. The dataset were all documents were concatenated (Total), scores the worst. The dataset containing information on the university from the Wikipedia page, scores the best.

Surprising is the fact that the Frog algorithm actually performs well on the datasets were the number of unknown words to the CELEX database (1 and 3), was the largest. This high performance suggests that the Frog algorithm, that gets activated if a word is unknown, performs very well.

The overall best scoring method was the BT algorithm using the manual tags to stem the words. This was to be expected since the BT algorithm was designed to stem a word to a stem, that corresponds with the manual stem.

Dataset	1	2	3	4	Total
Porter	0.9254	0.9330	0.9692	0.9382	0.9066
Lancaster	0.9300	0.9489	0.9822	0.9382	0.9262
Snowball	0.9348	0.9543	0.9736	0.9382	0.9276
BT + Frog tag	0.9789	0.9766	0.9910	0.9766	0.9678
BT + Manual tag	0.9821	9.9806	0.9930	0.9783	0.9854
BT + model tag	0.9673	0.9766	0.9764	0.9766	0.9656

Table 27. Understemming of the algorithms on evaluation datasets

Table 28 shows similar results as in Table 27, although the differences are smaller. The Porter algorithm in particular shows high accuracy in overstemming. The BT algorithm performs slightly less than the Frog tagger, but the reason for this is the almost brute-force and sophisticated algorithm for unknown words from the Frog tagger.

Dataset	1	2	3	4	Total
Porter	0.9848	0.9771	0.9912	0.9848	0.9753
Lancaster	0.9744	0.9708	0.9583	0.9744	0.9482
Snowball	0.9795	0.9884	0.9821	0.9795	0.9698
BT + Frog tag	0.9894	0.9882	0.9729	0.9894	0.9690
BT + Manual tag	0.9832	0.9813	0.9798	0.9921	0.9722
BT + model tag	0.9855	0.9876	0.9729	0.9898	0.9654

Table 28. Overstemming of the algorithms on evaluation datasets

The third performance measure was computational speed. Table 29 only takes into account to computation time needed to stem the words after a tag was determined by the tagging

module.

An important note while considering the computation times is that Porter, Lancaster and Snowball have been written in C, which may be considered a low-level programming language C compared to Python. The BT algorithm, on the other hand, was built in Python 3.6 which is a higher-level programming language and therefore requires more interpretation software layers for execution. Some optimisation was done in the programming but there is still room for improvement. Programmes written in C are usually much faster than programmes written in Python (Oliphant, 2007). A comparison study (Prechelt, 2000) among various programming languages showed that the average computation time could be improved by factor 2 or 3, whereas the memory consumption would also decrease by roughly the same amount.

Table 29 shows that the BT algorithm is considerably slower, due to one of the reasons previously mentioned, but is still able to stem a single document in under one second (1000ms).

Dataset	1	2	3	4	Total
Porter	9.21 (2.04)	9.15 (3.13)	10.60 (2.22)	8.44 (2.84)	30.60 (4.71)
Lancaster	9.10 (2.21)	7.83 (1.66)	10.80 (2.30)	7.46 (1.74)	29.40 (3.27)
Snowball	6.20 (1.35)	5.67 (1.54)	7.67 (2.39)	5.74 (1.43)	20.60 (3.76)
BT	737.14 (18.63)	702.84 (17.25)	754.18 (19.31)	676.38 (17.45)	2,580 (65.90)

Table 29. Computational time of stemmers in ms: mean (std. dev.) of 100 runs, 100 loops each

Table 30 shows the difference in computation time on determining a tag between the Frog algorithm and the BT algorithm. Again an important not in interpreting Table 30 is that Frog runs on a server. Therefore the computation speed is not only determined by how fast the algorithm is able to process the information, but also the internet speed of the user. Then again the BT algorithm is programmed in Python 3.6, while it is unknown in which language Frog is build, presumably C.

Before a file can be stemmed there are some preprocessing steps required and these are taken into account when Table 30 is examined.

Dataset	1	2	3	4	Total
Frog	79.32 (3.23)	49.81 (1.10)	53.12 (4.14)	47.29 (0.98)	136.84 (7.38)
BT	2.83 (0.21)	1.79 (0.14)	2.11 (0.18)	1.64 (0.17)	6.21 (0.92)

Table 30. Computational time of Taggers in seconds: mean (std. dev.) of 100 runs, 100 loops each

As the Frog tags were considered the ground truth during the training on the various models a comparison is made between the Manual tags and the Frog tags. The confusion matrix of this comparison is presented is Table 31, with the Frog F_1 scores presented in Table 32.

Frog \ Manual	ZNW	BVNW	BW	ovt	ott	vtt	Total
ZNW	273	17	2	5	2	0	299
BVNW	5	131	0	1	0	2	139
BW	65	30	454	4	2	0	555
ovt	0	6	0	36	0	0	42
ott	10	2	0	0	154	3	169
vtt	3	1	0	2	0	18	24
Total	356	187	456	48	158	23	1228

Table 31. Confusion matrix Frog and manual tags

Tag	F_1 - Score
ZNW	0.8235
BVNW	0.8037
BW	0.8447
ovt	0.8000
ott	0.9249
vtt	0.7660

Table 32. F_1-scores of the Frog tagging algorithm on the manual dataset

5 CONCLUSION & DISCUSSION

5.1 Conclusion

Out of the three algorithms - *MLR*, *NN*, *XGB* - that were tested as a possibility for a new tagger, XGB was shown to be the most accurate. Furthermore, the algorithm that distinguished between the verb categories made use of XGB, due to the resulting high F_1-scores. On the one hand, this is a surprising result, since no other instances of XGB being used as a tagging algorithm were found in the literature, despite the fact that it showed promise with other classification problems.

The number of rules that are implemented in the BT algorithm is relatively small, compared to the current state-of-the-art stemming algorithms (i.e. Porter, Lancaster, Snowball). This is mainly due to the fact that the words follow tag-specific rules, obtained from the tagging algorithm.

Based on the results as discussed in Chapter 4 it seems that the new BT algorithm works for the stemming and tagging of words. The goal of this thesis was to build a new stemmer that would be more accurate than the state-of-the-art stemmers (i.e. Porter, Lancaster and Snowball), as well as faster than the Frog algorithm. This goal has been achieved, while still leaving room for a high degree of technical optimisation. The combination of the XGB model and new stemming rules manages to perform well with the manually tagged and stemmed database in terms of under- and overstemming, as well as computation speed, and thereby resulted in the new BT algorithm.

As can be expected, the Frog algorithm performed its tasks slower in the case that it encountered words that are not included in the CELEX database (i.e. dataset 1 on the BFG). Surprisingly, though, its resulting accuracy was fairly high, in some cases even performing better than situations wherein words could be found in the CELEX database.

The newly-built BT algorithm is not yet completely optimised in terms of programming and its ruleset. Programming it in a language that is faster than Python 3.6, such as C, should increase the computation speed. Additional experiments should be conducted on

larger datasets to see whether the stemmer rules need tuning for the accuracy to be further improved.

The overall speed of the BT algorithm is significantly higher than the current Frog algorithm. Combining this with the fact its performance with under- and overstemming was slightly worse than when using the Frog algorithm, yet still better than the truncating algorithms, suggest that this new development can be considered a preliminary success. The entire process of inserting PDF documents into the algorithm, preprocessing, then tagging and stemming them takes an estimated time of around 8 to 10 seconds. From extensive literature research, it can be inferred that this time can be improved by factor of 2 or 3.

Finally, after having tested the performance of the Frog algorithm, its performance turned out to be rather lacking, especially because it is considered the ground truth for the training models. The F_1 scores combined with the confusion matrix from Tables 31 and 32 show that 65 nouns were predicted to be an adverb, which suggests that the BT algorithm would not have stemmed these nouns.

5.2 Discussion

The new tagging module only differentiates between six possible tags, while there are more that could be stemmed. For example the various pronouns can be stemmed back to their respective personal pronouns. The numerical word types can also be reduced to one of the overarching numerical word types. However, this changes the meaning of the word which is not always preferable and are thereby tagged as an adverb, and are therefore not stemmed.

The evaluation dataset was rather small and only provides an indication of the performance of the various algorithms. Extending the dataset is a rather tedious job which requires a lot of time and concentration, as mistakes have to be avoided at all costs.

Like any other machine learning model the XGB-classifier can also be enhanced further by adding more training data. By expanding the training set with a larger variety of words of which their correct tag is known in advance, XGB can make more accurate predictions. The number of optimised parameters can also be expanded, but due to the long training time of

this model the number of parameters was limited to those described in Section 2.3.3.

Unfortunately, due to time restrictions, the analysis on feature importance (i.e. which features imply a certain tag) could not be conducted. This will be done in the near future, when the Bag & Tag'em algorithm is ready to be distributed.

With the promising results presented in this thesis and a clear path for the future, the goal of building a new stemming algorithm that is more accurate than the current state-of-the-art stemmers, but faster than the Frog algorithm, has been achieved.

5.3 Further Research

Since there was no available data that was pre-tagged and included full sentences, sequential models could not yet be implemented. Despite the encouraging results achieved in this thesis with respect to under- and overstemming, this could be the logical next step as the literature suggests that this could enhance the tagging performance, and thereby stemming algorithms. The sequential model should not replace the current tagging module but should be seen as a source of additional correction after a word is tagged by the tagger.

One suggestion would be to make use of anchor points that are provided to the tagger in order to assist the sequential model. Anchor points in a sentence would be very common words: personal forms, articles, forms of the most used verbs (to have, to shall, to do, to will). In addition, this could be combined with a simple heuristic that would recognise a capital letter in the middle of a sentence as a name.

The sentence information can be used in a pervasive attention model which would not only look at sequences but also anchor points, as described. Combining the predicted tag with a prefix and suffix model would further enhance the probability of selecting the correct tag.

Further analysis on feature importance has to be conducted to determine whether certain features should be excluded from the BT algorithm to improve the XGB model, due to the possibility of overfitting.

Finally, as previously mentioned in Section 5.1, the programming language that the BT algorithm is written in should be altered. This has not been implemented yet due to the fact that the author only had sufficient programming experience with Python and not with a low-level programming languages, such as C.

Bibliography

Aizawa, A. (2003). An information-theoretic perspective of tf–idf measures. *Information Processing & Management*, *39*(1), 45–65.

Ajax, P. (2019, July 2). Ajax is op zoek naar een Assistent-Controller. Retrieved July 3, 2019, from https://www.ajax.nl/club/vacatures.htm

Berry, K. J. & Mielke Jr, P. W. (1988). A generalization of Cohen's kappa agreement measure to interval measurement and multiple raters. *Educational and Psychological Measurement*, *48*(4), 921–933.

Bosch, A. v. d., Busser, B., Canisius, S., & Daelemans, W. (2007). An efficient memory-based morphosyntactic tagger and parser for Dutch. *LOT Occasional Series*, *7*, 191–206.

Breiman, L. (1996). Bagging predictors. *Machine learning*, *24*(2), 123–140.

Brill, E. (1992). A simple rule-based part of speech tagger. In *Proceedings of the third conference on Applied natural language processing* (pp. 152–155). Association for Computational Linguistics.

Cavnar, W. B., Trenkle, J. M. et al. (1994). N-gram-based text categorization. In *Proceedings of SDAIR-94, 3rd annual symposium on document analysis and information retrieval* (Vol. 161175). Citeseer.

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, *16*, 321–357.

Chen, T. & Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* (pp. 785–794). ACM.

Chinchor, N. (1992). MUC-4 evaluation metrics. In *Proceedings of the 4th conference on Message understanding* (pp. 22–29). Association for Computational Linguistics.

Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Clevert, D.-A., Unterthiner, T., & Hochreiter, S. (2015). Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*.

Drummond, C., Holte, R. C. et al. (2003). C4. 5, class imbalance, and cost sensitivity: why under-sampling beats over-sampling. In *Workshop on learning from imbalanced datasets II* (Vol. 11, pp. 1–8). Citeseer.

Elbayad, M., Besacier, L., & Verbeek, J. (2018). Pervasive attention: 2d convolutional neural networks for sequence-to-sequence prediction. *arXiv preprint arXiv:1808.03867*.

Feldman, R. & Sanger, J. (2007). *The text mining handbook: advanced approaches in analyzing unstructured data*. Cambridge university press.

Freund, Y., Schapire, R., & Abe, N. (1999). A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, *14*(771-780), 1612.

Friedman, J. H. (2002). Stochastic gradient boosting. *Computational statistics & data analysis*, *38*(4), 367–378.

Fu, W. J. (1998). Penalized regressions: the bridge versus the lasso. *Journal of computational and graphical statistics*, *7*(3), 397–416.

Gautam, K. (2018). Ensemble Methods: Boosting. *CS6375: Machine Learning*, 14.

Gilroy, E., Hirsch, R., & Cohn, T. (1990). Mean square error of regression-based constituent transport estimates. *Water Resources Research*, *26*(9), 2069–2077.

Greene, W. H. (2003). *Econometric analysis*. Pearson Education India.

Haeseryn, W. J.-M., Romijn, K., Geerts, G., Rooij, J. d., & Van den Toorn, M. C. (1997). Algemene Nederlandse Spraakkunst [2 banden].

Hand, D. & Christen, P. (2018). A note on using the F-measure for evaluating record linkage algorithms. *Statistics and Computing*, *28*(3), 539–547.

Hansen, L. K. & Salamon, P. (1990). Neural network ensembles. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (10), 993–1001.

Hastie, T., Tibshirani, R., Friedman, J., & Franklin, J. (2005). The elements of statistical learning: data mining, inference and prediction. *The Mathematical Intelligencer*, *27*(2), 120–122.

He, J. & Zelikovsky, A. (2006). MLR-tagging: informative SNP selection for unphased genotypes based on multiple linear regression. *Bioinformatics*, *22*(20), 2558–2561.

Hinton, G. E. (1987). Learning translation invariant recognition in a massively parallel networks. In *International Conference on Parallel Architectures and Languages Europe* (pp. 1–13). Springer.

Ho, T. K. (1995). Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition* (Vol. 1, pp. 278–282). IEEE.

Hochreiter, S. (1998). The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, *6*(02), 107–116.

Jivani, A. G. et al. (2011). A comparative study of stemming algorithms. *Int. J. Comp. Tech. Appl*, *2*(6), 1930–1938.

Kontostathis, A., Edwards, L., & Leatherman, A. (2010). Text mining and cybercrime. *Text Mining: Applications and Theory. John Wiley & Sons, Ltd, Chichester, UK*, 149–164.

Krizhevsky, A. & Hinton, G. (2010). Convolutional deep belief networks on cifar-10. *Unpublished manuscript*, *40*(7), 1–9.

Marcelis, H. (2019, June 13). Groningen getroffen door vierde aardbeving in twee weken tijd. *Algemeen Dagblad*. Retrieved June 13, 2019, from https://www.ad.nl/binnenland/groningen-getroffen-door-vierde-aardbeving-in-twee-weken-tijd~a5a76dce/?referrer=https://www.google.com/

Mattmann, C. & Zitting, J. (2011). *Tika in action*. Manning Publications Co.

Mijnwoordenboek.nl. (2004). Mijnwoordenboek werkwoorden. Retrieved March 15, 2019, from https://www.mijnwoordenboek.nl/werkwoorden/NL/

Monz, C. & De Rijke, M. (2001). Shallow morphological analysis in monolingual information retrieval for Dutch, German, and Italian. In *Workshop of the Cross-Language Evaluation Forum for European Languages* (pp. 262–277). Springer.

Moral, C., de Antonio, A., Imbert, R., & Ramırez, J. (2014). A survey of stemming algorithms in information retrieval. *Information Research: An International Electronic Journal*, *19*(1), n1.

Nielsen, M. A. (2015). *Neural networks and deep learning*. Determination press San Francisco, CA, USA:

Oliphant, T. E. (2007). Python for scientific computing. *Computing in Science & Engineering*, *9*(3), 10–20.

Olson, D. L. & Delen, D. (2008). *Advanced data mining techniques*. Springer Science & Business Media.

Paice, C. D. (1990). Another stemmer. In *ACM Sigir Forum* (Vol. 24, *3*, pp. 56–61). ACM.

Paik, J. H. (2013). A novel TF-IDF weighting scheme for effective ranking. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval* (pp. 343–352). ACM.

Pang, B., Lee, L. et al. (2008). Opinion mining and sentiment analysis. *Foundations and Trends® in Information Retrieval*, *2*(1–2), 1–135.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Phua, C., Lee, V., Smith, K., & Gayler, R. (2010). A comprehensive survey of data mining-based fraud detection research. *arXiv preprint arXiv:1009.6119*.

Porter, M. F. (2001). Snowball: A language for stemming algorithms.

Prechelt, L. (2000). An empirical comparison of c, c++, java, perl, python, rexx and tcl. *IEEE Computer*, *33*(10), 23–29.

Rahul Bhatia. (2018). When not to use Neural Networks. [Online; accessed Juli 11, 2019]. Retrieved from https://medium.com/datadriveninvestor/when-not-to-use-neural-networks-89fb50622429

Rikunert. (2017). SMOTE explained for noobs - Synthetic Minority Over-sampling TEchnique line by line. Retrieved March 15, 2019, from http://rikunert.com/SMOTE_explained

Salton, G. & Buckley, C. (1988). Term-weighting approaches in automatic text retrieval. *Information processing & management*, *24*(5), 513–523.

Schmid, H. (1994). Part-of-speech tagging with neural networks. In *Proceedings of the 15th conference on Computational linguistics-Volume 1* (pp. 172–176). Association for Computational Linguistics.

Schmidt, M., Fung, G., & Rosales, R. (2007). Fast optimization methods for l1 regularization: A comparative study and two new approaches. In *European Conference on Machine Learning* (pp. 286–297). Springer.

Smith, R. (2007). An overview of the Tesseract OCR engine. In *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)* (Vol. 2, pp. 629–633). IEEE.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, *58*(1), 267–288.

Trottier, L., Gigu, P., Chaib-draa, B., et al. (2017). Parametric exponential linear unit for deep convolutional neural networks. In *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)* (pp. 207–214). IEEE.

Van der Beek, L., Bouma, G., Malouf, R., & Van Noord, G. (2002). The Alpino dependency treebank. In *Computational linguistics in the netherlands 2001* (pp. 8–22). Brill Rodopi.

Van der Wouden, T. (1990). Celex: Building a multifunctional polytheoretical lexical data base. *Proceedings of BudaLex*, *88*, 363–373.

van Gompel, M. & Hendrickx, I. (2019). LaMachine: A meta-distribution for NLP software. In *Selected papers from the CLARIN Annual Conference 2018, Pisa, 8-10 October 2018* (159, pp. 209–221). Linköping University Electronic Press.

Weiss, S. M., Indurkhya, N., & Zhang, T. (2015). *Fundamentals of predictive text mining*. Springer.

Weiss, S. M., Indurkhya, N., Zhang, T., & Damerau, F. (2010). *Text mining: predictive methods for analyzing unstructured information*. Springer Science & Business Media.

Vijayarani, S., Ilamathi, M. J., & Nithya, M. (2015). Preprocessing techniques for text mining-an overview. *International Journal of Computer Science & Communication Networks*, *5*(1), 7–16.

Vijayarani, S., Janani, M. R. et al. (2016). Text mining: open source tokenization tools-an analysis. *Advanced Computational Intelligence: An International Journal (ACII)*, *3*(1), 37–47.

Wikipedia, W. (2019, February 19). Vrije Universiteit Amsterdam. Retrieved June 13, 2019, from https://nl.wikipedia.org/wiki/Vrije_Universiteit_Amsterdam

Willett, P. (2006). The Porter stemming algorithm: then and now. *Program*, *40*(3), 219–223.

Volpi, M. & Tuia, D. (2016). Dense semantic labeling of subdecimeter resolution images with convolutional neural networks. *IEEE Transactions on Geoscience and Remote Sensing*, *55*(2), 881–893.

Xu, B., Wang, N., Chen, T., & Li, M. (2015). Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*.

Zhu, C., Byrd, R. H., Lu, P., & Nocedal, J. (1997). Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on Mathematical Software (TOMS)*, *23*(4), 550–560.

Zou, H. & Hastie, T. (2005). Regularization and variable selection via the elastic net. *Journal of the royal statistical society: series B (statistical methodology)*, *67*(2), 301–320.

Appendices

A Overview word types *taalkundig ontleden* in the Dutch Language

Type	Translation	Example	Translation
Lidwoord	Article	De lange vrouw	The tall woman
Zelfstandig naamwoord	Noun	De lange vrouw	The tall woman
Bijvoeglijk naamwoord	Adjective	De lange vrouw	The tall woman
Hulpwerkwoord	Auxiliary verb	Ik zal lopen	I shall walk
Koppelwerkwoord	Linking verb	Zij is een vrouw	She is a woman
Zelfstandig werkwoord	Independent verb	Vandaag ben ik ziek	Today I am ill
Persoonlijk voornaamwoord	Personal pronoun	Ik zal lopen	I shall walk
Bezittelijk voornaamwoord	Possessive pronoun	Mijn hond is blij	My dog is happy
Aanwijzend voornaamwoord	Demonstrative pronoun	Die hond is blij	That dog is happy
Betrekkelijk voornaamwoord	Relative pronoun	De hond die Henk heet	The dog whose name is Henk
Vragend voornaamwoord	Interrogative pronoun	Hoe heet de hond?	What is the dogs name?
Bepaald hoofdtelwoord	Determined numeral	Er zijn twee honden	There are two dogs
Onbepaald hoofdtelwoord	Undetermined numeral	Er zijn weinig honden	There are a few dogs
Bepaald rangtelwoord	Determined ordinal numeral	De tweede hond	The second dog
Onbepaald rangtelwoord	Undetermined ordinal numeral	De laatste hond	The last dog
Onbepaald voornaamwoord	Indefinite pronoun	De andere hond	The other dog
Wederkerend voornaamwoord	Recurring pronoun	De hond schaamt zich	The dog is ashamed of himself
Wederkerig voornaamwoord	Reciprocal pronoun	De honden wassen elkaar	The dogs clean each other
Voorzetsel	Preposition	Ik geef dit aan haar	I give this to her
Bijwoord	Adverb	Morgen komt hij niet	He will not come tomorrow
Voegwoord	Conjunction	De hond blaft en ik slaap	The dog barcs and I am asleep
Tussenwerpsel	Interjection	Foei! stoute hond	Shame! bad dog

Table 33. Overview all word types *taalkundig ontleden* in the Dutch Language

B Overview forms of times in Dutch language

Form of time	Dutch example	Translation	Translation example
Onvoltooid tegenwoordige tijd (ott)	ik leer	Present tense	I study
Voltooid tegenwoordige tijd (vtt)	ik heb geleerd	Present perfect	I have studied
Onvoltooid verleden tijd (ovt)	ik leerde	Simple past tense	I studied
Voltooid verleden tijd (vvt)	ik had geleerd	Past perfect tense	I had studied
Onvoltooid tegenwoordige toekomende tijd (ottt)	ik zal leren	Simple present Future tense	I shall study
Voltooid tegenwoordige toekomende tijd (vttt)	ik zal geleerd hebben	Complete present future tense	I shall have studied
Onvoltooid verleden toekomende tijd (ovtt)	ik zou leren	Unfinished past tense	I should have studied
Voltooid verleden toekomende tijd (vvtt)	ik zou geleerd hebben	Completed past future tense	I should have studied
Voltooid deelwoord (VD)	geleerd	Past participle	studied
Gebiedende wijs	leer	Imperative	study!

Table 34. Verb forms of time in Dutch language

C Code for the stemming algorithm

```
import pandas as pd

klinker = set(['a', 'e', 'o'])
special_klinker = set(['i', 'u'])


def REST_Stemmer(REST_data):
    REST_data['Final stem rest'] = REST_data['Word']
    REST_data = REST_data.drop(['BT tag', 'Alg stem'], axis=1)
    REST_data.to_excel(filename)


def BVNW_stemmer(BVNW_data):
    BVNW_data['Alg stem'] = BVNW_data['Word']
    BVNW_data.loc[BVNW_data['Alg stem'].str.endswith('er'), 'Alg stem'] = BVNW_data['Alg stem'].str[:-2]
    BVNW_data.loc[BVNW_data['Alg stem'].str.endswith('ste'), 'Alg stem'] = BVNW_data['Alg stem'].str[:-3]
    BVNW_data.loc[BVNW_data['Alg stem'].str.endswith('ende'), 'Alg stem'] = BVNW_data['Alg stem'].str[:-4]
    BVNW_data.loc[BVNW_data['Alg stem'].str.endswith('en'), 'Alg stem'] = BVNW_data['Alg stem'].str[:-2]
    BVNW_data.loc[BVNW_data['Alg stem'].str.endswith('e'), 'Alg stem'] = BVNW_data['Alg stem'].str[:-1]

    # Dirty f and s
    BVNW_data.loc[BVNW_data['Alg stem'].str.endswith('v'), 'Alg stem'] = BVNW_data['Alg stem'].str.replace('v','f',-1)
    BVNW_data.loc[BVNW_data['Alg stem'].str.endswith('z'), 'Alg stem'] = BVNW_data['Alg stem'].str.replace('z','s',-1)

#    # Doubling of a,e,o,u
    BVNW_data.loc[BVNW_data['Alg stem'].str[-2].str.match('a') & (~BVNW_data['Alg stem'].str[-3].isin(klinker)) &
    (~BVNW_data['Alg stem'].str.endswith('r')), 'Alg stem'] = BVNW_data['Alg stem'].str.replace('a','aa',-1)
    BVNW_data.loc[BVNW_data['Alg stem'].str[-2].str.match('e') & (~BVNW_data['Alg stem'].str[-3].isin(klinker)) &
    (~BVNW_data['Alg stem'].str.endswith('r')), 'Alg stem'] = BVNW_data['Alg stem'].str.replace('e','ee',-1)
    BVNW_data.loc[BVNW_data['Alg stem'].str[-2].str.match('o') & (~BVNW_data['Alg stem'].str[-3].isin(klinker)) &
    (~BVNW_data['Alg stem'].str.endswith('r')), 'Alg stem'] = BVNW_data['Alg stem'].str.replace('o','oo',-1)
    BVNW_data.loc[BVNW_data['Alg stem'].str[-2].str.match('u') & (~BVNW_data['Alg stem'].str[-3].isin(klinker)) &
    (~BVNW_data['Alg stem'].str[-3].isin(special_klinker)), 'Alg stem'] = BVNW_data['Alg stem'].str.replace('u','uu',-1)

#    # double letter at the end
    BVNW_data.loc[BVNW_data['Alg stem'].str[-1] == BVNW_data['Alg stem'].str[-2], 'Alg stem'] = BVNW_data['Alg stem'].str[:-1]
    BVNW_data['Final stem BVNW'] = BVNW_data['Alg stem']
    BVNW_data = BVNW_data.drop(['BT tag','Alg stem'], axis=1)


    BVNW_data.to_excel(filename)


def ZNW_Stemmer(ZNW_data):

        # Preparing the data

        ZNW_data['Alg stem'] = ZNW_data['Word']
        ZNW_data['BT_plural'] = 'ev'
        ZNW_data.loc[ZNW_data['Alg stem'].str.endswith('en'), 'BT_plural'] = ZNW_data['BT_plural'].replace('ev', 'mv')
        ZNW_data.loc[ZNW_data['Alg stem'].str.endswith('jes'), 'BT_plural'] = ZNW_data['BT_plural'].replace('ev', 'mv')
        ZNW_data.loc[ZNW_data['Alg stem'].str.endswith('s'), 'BT_plural'] = ZNW_data['BT_plural'].replace('ev', 'mv')
        ZNW_data.loc[ZNW_data['Alg stem'].str.endswith('en') & (ZNW_data['BT_plural'].str.match('mv')), 'Alg stem'] =
        ZNW_data['Alg stem'].str[:-2]
        ZNW_data.loc[ZNW_data['Alg stem'].str.endswith('jes') & (ZNW_data['BT_plural'].str.match('mv')), 'Alg stem'] =
        ZNW_data['Alg stem'].str[:-3]
        ZNW_data.loc[ZNW_data['Alg stem'].str.endswith('jes') & (ZNW_data['BT_plural'].str.match('mv')), 'Alg stem'] =
        ZNW_data['Alg stem'].str[:-1]
        #Foul f and s
        ZNW_data.loc[ZNW_data['Alg stem'].str.endswith('v') & (ZNW_data['BT_plural'].str.match('mv')), 'Alg stem'] =
        ZNW_data['Alg stem'].str.replace('v','f',-1)
```

```
        ZNW_data.loc[ZNW_data['Alg stem'].str.endswith('z') & (ZNW_data['BT_plural'].str.match('mv')), 'Alg stem'] =
        ZNW_data['Alg stem'].str.replace('z','s',-1)
        # double letter at the end
        ZNW_data.loc[ZNW_data['Alg stem'].str[-1] == ZNW_data['Alg stem'].str[-2], 'Alg stem'] =
        ZNW_data['Alg stem'].str[:-1]
        ZNW_data['Final stem ZNW'] = ZNW_data['Alg stem']
        ZNW_data = ZNW_data.drop(['BT tag','Alg stem', 'BT_plural'], axis=1)

        ZNW_data.to_excel(filename)




def WW_Stemmer(WW_data):
    # Preparing the data
    WW_data['FirstChars'] = [x[:3] for x in WW_data['Word']]
    WW_data['Samenstelling'] = pd.np.where(WW_data.FirstChars.str.match("aan"), "aan",
                    pd.np.where(WW_data.FirstChars.str.match("af"), "af",
                    pd.np.where(WW_data.FirstChars.str.match("bij"), "bij",
                    pd.np.where(WW_data.FirstChars.str.match("in"), "in",
                    pd.np.where(WW_data.FirstChars.str.match("op"), "op",
                    pd.np.where(WW_data.FirstChars.str.match("uit"), "uit",""))))))
    WW_data['Process'] = [e.replace(k, '',1) for e, k in zip(WW_data.Word.astype('str'),
    WW_data.Samenstelling.astype('str'))]
    WW_data = WW_data.drop(['FirstChars'], axis=1)
    irregular = pd.read_excel(filename)
    WW_data = pd.merge(WW_data, irregular, on='Process', how= 'left')
    WW_data['BT Irregular'] = 1
    WW_data['Irregular Stem'] = WW_data['Irregular Stem'].fillna(' ')
    WW_data.loc[WW_data['Irregular Stem'].str.contains(' '), 'BT Irregular'] = WW_data['BT Irregular'].replace(1, 0)
    WW_data['Alg stem'] = WW_data['Process']
    WW_data['BT_plural'] = 'ev'


    # To ensure BVNW is seen as mv
    WW_data.loc[WW_data['Alg stem'].str.endswith('en'), 'BT_plural'] = WW_data['BT_plural'].replace('ev', 'mv')


    # VTT/ VVT
    WW_data.loc[WW_data['BT tag'].str.match('vtt') & (WW_data['Alg stem'].str.startswith('ge')), 'Alg stem'] =
    WW_data['Alg stem'].str[2:]
    WW_data.loc[WW_data['BT tag'].str.match('vtt') & (WW_data['Alg stem'].str.endswith('d')), 'Alg stem'] =
    WW_data['Alg stem'].str[:1]

    # GENERAL (OTT)
    # Remove -en but ensure lentgh is at least 3 chars long
    WW_data.loc[WW_data['Alg stem'].str.endswith('en') & (WW_data['Alg stem'].str.len() > 3), 'Alg stem'] =
    WW_data['Alg stem'].str[:-2]
    # Converts ovt ev to ott ev
    WW_data.loc[WW_data['BT tag'].str.match('ovt') & (WW_data['Alg stem'].str.endswith('e')) &
    (WW_data['Alg stem'].str.len() > 3), 'Alg stem'] = WW_data['Alg stem'].str[:-1]
    # Remove t after dt
    WW_data.loc[WW_data['Alg stem'].str.endswith('t') & (WW_data['Alg stem'].str[-2] == 'd'), 'Alg stem'] =
    WW_data['Alg stem'].str[:-1]
    # If not irregular remove t at the end
    WW_data.loc[WW_data['Alg stem'].str.endswith('t') & (WW_data['BT Irregular'] == 0)  &
    (~WW_data['Alg stem'].str[-2].isin(klinker)), 'Alg stem'] = WW_data['Alg stem'].str[:-1]
    # Foul s and f
    WW_data.loc[WW_data['Alg stem'].str.endswith('v'), 'Alg stem'] =
    WW_data['Alg stem'].str.replace('v','f',-1)
    WW_data.loc[WW_data['Alg stem'].str.endswith('z'), 'Alg stem'] =
    WW_data['Alg stem'].str.replace('z','s',-1)
```

```
    # Doubling of vowels
    WW_data['Alg_stem_revers'] = WW_data.loc[:,'Alg_stem'].apply(lambda x: x[::-1])
    WW_data.loc[WW_data['Alg_stem_revers'].str[2].isin(klinker) & (~WW_data['Alg_stem_revers'].str[3].isin(klinker))
    & (WW_data['BT_plural'].str.match('mv')) & (WW_data['Alg_stem'].str[-2].str.match('a')), 'Alg_stem_revers'] =
    WW_data['Alg_stem_revers'].str.replace('a','aa',-1)
    WW_data.loc[WW_data['Alg_stem_revers'].str[2].isin(klinker) &
    (~WW_data['Alg_stem_revers'].str[3].isin(klinker)) & (WW_data['BT_plural'].str.match('mv')) &
    (WW_data['Alg_stem'].str[-2].str.match('e')), 'Alg_stem_revers'] = WW_data['Alg_stem_revers'].str.replace('e','ee',-1)
    WW_data.loc[WW_data['Alg_stem_revers'].str[2].isin(klinker) &
    (~WW_data['Alg_stem_revers'].str[3].isin(klinker)) & (WW_data['BT_plural'].str.match('mv')) &
    (WW_data['Alg_stem'].str[-2].str.match('o')), 'Alg_stem_revers'] =
    WW_data['Alg_stem_revers'].str.replace('o','oo',-1)
    WW_data['Alg_stem'] = WW_data.loc[:,'Alg_stem_revers'].apply(lambda x: x[::-1])



    #    This is for correcting the double letters (Becarefull for slee)
    WW_data.loc[WW_data['Alg_stem'].str[-1] == WW_data['Alg_stem'].str[-2], 'Alg_stem'] = WW_data['Alg_stem'].str[:-1]

#    This is for handling irregular verbs
    WW_data.loc[WW_data['BT_Irregular'] == 1, 'Alg_stem'] = WW_data['Irregular_Stem']

#    This is for finalising the stem
    WW_data['Final_stem_WW'] = WW_data['Samenstelling'].fillna('') + WW_data['Alg_stem']
    WW_data = WW_data.drop(['BT_tag', 'Alg_stem', 'Samenstelling', 'Process', 'Irregular_Stem',
    'BT_Irregular', 'BT_plural'], axis=1)

    WW_data.to_excel(filename)

if __name__== "__main__":
    data = pd.read_excel(filename)
    data['Alg_stem'] = data['Word']
    WW_data = data.loc[data['BT_tag'].str.match(pat = '(ott)|(ovt)|(vtt)')]
    WW_Stemmer(WW_data)
    BVNW_data = data.loc[data['BT_tag'].str.match('BVNW')]
    BVNW_stemmer(BVNW_data)
    ZNW_data = data.loc[data['BT_tag'].str.match('ZNW')]
    ZNW_Stemmer(ZNW_data)
    REST_data = data.loc[~data['BT_tag'].str.match(pat = '(ott)|(ovt)|(vtt)|(BVNW)|(ZNW)')]
    REST_Stemmer(REST_data)
```

D Exception list

Due to the size of the exception list, this list is not included. However the exception list will be made publicly available when the Bag & Tag'em algorithm becomes available online. If one wants to see the exception list before hand please contact the Author.

E Evaluation documents

This part of the Appendix contains the four text documents used for evaluation purposes which were stemmed and tagged manually.

E.1 Document 1 (BFG)

Snoskommers Het weeskind Sofie ziet op zekere nacht hoe een reus door de straat stapt en door een lange trompet uit een glazen pot dromen blaast in de slaapkamers van de kinderen. De reus merkt dat Sofie hem ziet en "als iemand ooit een reus ziet, moet hij of zij hupsakadee weggehaald worden." Sofie is natuurlijk vreselijk bang als de reus haar ontvoert. Ze verwacht elk ogenblik dat hij haar zal opeten. Maar dan blijkt dat deze reus heel anders is dan al de verschrikkelijke bloeddorstige reuzen - de Vleeslapeter, de Bottenkraker, de Mensenmepper, de Kinderkauwer, de Vleeshakker, de Schrokschranzer, de Meisjesstamper, de Bloedbottelaar en de Slagersjongen ... Dit is namelijk de GVR, de Grote Vriendelijke Reus. Grote Vriendelijke Reus "Maar als jij dan geen mensen eet, zoals de anderen", zei Sofie, "waar leef je dan van?" "Dat is een alledeksels moeilijk probleem hier", antwoordde de GVR, "In dit beslabberde Reuzenland groeit gewoonweg geen vrolijke vruchten zoals anananananassen en lachebekjes. Er groeit hier niks behalve een vreselijke akkiebakkie rotgroente. Hij heet snoskommer." "Snoskommer!" riep Sofie, "Die bestaat niet!" De GVR keek Sofie aan en glimlachte ongeveer twintig van zijn vierkante, witte tanden bloot. "Gisteren," zei hij, "gelooft we niet aan reuzen, hè? Vandaag gelooft we niet aan snoskommers. Alleen omdat we toevallig iets niet zelf gezien heeft met onze eigen koekeloertjes, denkt we dat het niet bestaat. Hoe zit het dan, om maar eens wat te noemen, met de grote kwistige winkelpoot?" "Wat zeg je?" vroeg Sofie "En met de grasgarnaal?" "Wat is dat?" vroeg Sofie. "En de trapkrabber?" "De wàt?" vroeg Sofie. "En de schurfsluiper?" "Zijn dat dieren?" vroeg Sofie? "Dat is de doodgewoonste dieren", zei de GVR minachtend, "Ik is zelf niet zo'n erge weet-al reus, maar het lijkt mij dat jij wel een heel erg weet-niks mensbaksel is. Jouw hoofd zit vol vaagsel." "Zaagsel, bedoel je", zei Sofie. "Wat ik bedoelt en wat ik zegt zijn twee verschillende dingen", zei de GVR uit de hoogte, "Nu zult ik jou een snoskommer laten zien." De GVR rukte een grote kast open en pakte het raarste ding dat Sofie ooit had gezien. Het was ongeveer half zo groot als een gemiddelde mens, maar veel dikker. In het midden was het zo breed als een kinderwagen. Het was zwart met witte strepen in de lengte. Er zaten ruwe knobbels aan.

E.2 Document 2 (Newspaper article)

Groningen getroffen door vierde aardbeving in twee weken tijd

Weer is het raak in Groningen: iets na 15.30 uur vanmiddag vond een aardbeving plaats met een kracht van 1,9 op de schaal van Richter. Het epicentrum lag in Appingedam. Het is in de provincie Groningen alweer de vierde beving in minder dan twee weken tijd.

Het KNMI meldt dat de beving vanmiddag om 15.37 uur plaatsvond in Appingedam. De beving had een kracht van 1,9 op de schaal van Richter. Eerdere bevingen deze maand in de provincie hadden een kracht van 0,6, 1,9 en 2,5. Die laatste vond plaats bij Garrelsweer.

De bevingen in Groningen worden veroorzaakt door de gaswinning in het gebied. Het KNMI laat weten dat zowel het aantal aardschokken als de kracht ervan in het afgelopen decennium aanzienlijk toenamen. De gaskraan wordt daarom geleidelijk dichtgedraaid. Vorig jaar telde het KNMI in totaal 113 aardbevingen in ons land. De meeste (87) deden zich voor in het aardgasgebied in Groningen.

Stuwmeerregeling

Voor de Groningers die door de beving vandaag schade hebben aan hun woning is de timing erg slecht. De deadline van de stuwmeerregeling is in de nacht van 12 juni namelijk verstreken. De regeling houdt in dat iedereen die een schademelding heeft gedaan bij de Tijdelijke Commissie Mijnbouwschade Groningen (TCMG) een vaste vergoeding van 5000 euro kan krijgen, zonder eerst te hoeven wachten op een beoordeling van de schade. Het is de bedoeling op deze manier een groot deel van de al lang openstaande schademeldingen (het 'stuwmeer') weg te werken. TCMG laat weten dat ze waarschijnlijk 15.000 van de 20.265 openstaande schademeldingen kan afhandelen met de regeling.

De mensen die vandaag schade hebben gekregen aan hun woning en dus net buiten de stuwmeerregeling vallen, hoeven niet te verwachten dat de deadline wordt verruimd. In een brief die minister Eric Wiebes (Economische Zaken en Klimaat) vandaag stuurde aan de Tweede Kamer onderkende hij al dat een eenmalige regeling met een harde grens altijd leidt tot een ongelijke behandeling tussen bestaande en nieuwe gevallen. Maar omdat de regeling er mede voor zorgt dat nieuwe schades sneller worden afgehandeld, acht de minister dat gerechtvaardigd.

E.3 Document 3 (Wikipedia article on VU)

De Vrije Universiteit Amsterdam (afgekort VU) is een brede onderzoeks- en onderwijsuniversiteit in Amsterdam. De VU is een van de twee universiteiten in Amsterdam, de andere is de Universiteit van Amsterdam (UvA). In 2017 telde de VU 22.867 studenten. De VU heeft een eigen academisch ziekenhuis, het VU medisch centrum (VUmc). Het VU-complex is gevestigd aan de De Boelelaan in Buitenveldert, nabij de Zuidas.

De VU is een bijzondere universiteit: zij gaat niet uit van de overheid, maar van de Stichting VU die de bestuurlijke koepel vormt boven de Vrije Universiteit. Deze stichting gaat op haar beurt uit van de VU-Vereniging, die in 1879 werd opgericht om de stichting van de VU mogelijk te maken. De VU werd gesticht als protestants-christelijke universiteit.[3]

De slogan van de universiteit is 'VU is verder kijken'. Het klassieke motto luidt: *Auxilium nostrum in nomine Domini*, ofwel: 'Onze hulp is in de naam des Heren', de woorden die ook ter opening van officiële bijeenkomsten (opening academisch jaar, *Dies Natalis*, promoties) gesproken worden.

De VU werd opgericht op 20 oktober 1880 door een groep orthodox-protestantse christenen onder leiding van de theoloog, journalist en politicus Abraham Kuyper (tevens oprichter van de Anti-Revolutionaire Partij en later, van 1901 tot 1905, premier). De oprichting van de VU had direct te maken met Kuypers streven naar een volwaardig bestaan voor de orthodox gereformeerden, de zogeheten *kleyne luyden*: zij konden nu studeren aan hun eigen universiteit.[4] Ook konden aan de VU predikanten worden opgeleid die calvinistisch in de leer waren, in tegenstelling tot predikanten die van de vrijzinnige theologische faculteiten aan de rijksuniversiteiten kwamen. Met de VU wilde Kuyper echter meer dan de kleine luiden emanciperen of orthodoxe predikanten opleiden. Hij wilde dat de academische wereld niet langer zou worden gedomineerd door de liberalen, die volgens hem een niet-christelijke, zogenaamd neutrale wetenschap beoefenden. Volgens Kuyper was het mogelijk een christelijke wetenschap te ontwikkelen op basis van de zogenaamde "gereformeerde beginselen". De door Kuyper en zijn medestanders ontwikkelde school van denken - op het terrein van de wetenschap, politiek en samenleving - wordt neocalvinisme genoemd.

E.4 Document 4 (Vacancy Ajax)

Ajax is op zoek naar een Assistent-Controller

AFC Ajax is de meest succesvolle voetbalclub van Nederland en heeft de ambitie om zowel in Nederland als in Europa tot de top van het professionele voetbal te behoren. Achter deze ambities en club staat een organisatie, waar, inclusief de spelers, een kleine 500 mensen werken.

Wie zoeken wij?
Als Assistent-Controller ondersteun je de financial- en business controller en geef je primair inhoud aan riskmanagement en de administratieve organisatie en interne beheersing. Daarnaast draag je zorg voor essentiële stuurinformatie voor de directie en het management.

Organisatie
De Assistent-Controller rapporteert aan de Manager Controlling. De afdeling Controlling maakt onderdeel uit van de portefeuille Financiële Zaken.

Wat ga je doen?

Financial controlling (primair)
- Mede verantwoordelijk voor het opzetten en vastleggen van een adequate administratieve organisatie en interne beheersing (AO/IB), inclusief proces- en procedurebeschrijvingen en richtlijnen
- Toetsen van de administratieve organisatie en interne beheersing (AO/IB) door het uitvoeren van interne audits
- Ondersteunen bij het opstellen en beoordelen op juistheid en volledigheid van externe rapportages naar de KNVB en UEFA
- Controleren op juistheid en volledigheid van de specificaties in het kader van de jaarrekening en halfjaarrekening
- Ondersteunen bij het opstellen van de jaarrekening
- Ontwikkeling en/of verbetering van de (geautomatiseerde) systemen en procedures ter optimalisatie van de financiële verslaggeving
- Ondersteunen bij het afhandelen van belastingissues
- Ondersteunen bij het beheer van de verzekeringsportefeuille

Business controlling (secundair)
- Ondersteunen bij het opstellen van begrotingen, prognoses en forecasts
- Ondersteunen bij het opstellen en beoordelen op juistheid en volledigheid van diverse periodieke rapportages waaronder de maandrapportages en dashboards
- Controleren en analyseren van de financiële gegevens en verklaren van fluctuaties

Wat neem je samengevat mee?
- HBO+ werk- en denkniveau, Bachelor (richting controlling, bedrijfseconomie of accountancy); voorkeur Master (richting controlling, bedrijfseconomie of accountancy)
- Relevante werkervaring (minimaal 3 jaar) bij een accountantskantoor of (middel)grote organisatie
- Ervaring met proces- en procedurebeschrijvingen en het opstellen van richtlijnen (AO/IB)
- Kennis van rapporteren onder IFRS en kennis van Afas/Profit is een pré
- Goede beheersing van de Nederlandse en Engelse taal in woord en geschrift
- Goede beheersing van Excel
- Goede analytische en communicatieve vaardigheden
- Sterke en daadkrachtige persoonlijkheid
- Stressbestendig, blijft onder druk goed presteren
- Flexibele instelling
- Integriteit

Wat hebben wij te bieden? Een fulltime baan in een sportieve en inspirerende omgeving waar iedere dag anders is en een marktconform salaris met goede secundaire arbeidsvoorwaarden.
De standplaats is de Johan Cruijff ArenA in Amsterdam.

Heb je interesse? Denk je dat jij de geschikte kandidaat bent en voldoe je aan het gevraagde profiel, dan kun je jouw motivatiebrief en up-to-date CV mailen naar: personeelszaken@ajax.nl.

acquisitie naar aanleiding van deze vacature wordt niet op prijs gesteld

Wij zijn Ajax.